



Edsger W. Dijkstra  
W.H.J. Feijen

***Een methode  
van  
programmeren***

ACADEMIC SERVICE



*Joh. S. Moench*

EEN METHODE VAN PROGRAMMEREN



Edsger W. Dijkstra  
W.H.J. Feijen

# ***Een methode van programmeren***

ACADEMIC SERVICE

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Dijkstra, Edsger W.

Een methode van programmeren / Edsger W. Dijkstra, W.H.J. Feijen.

- Den Haag : Academic Service

ISBN 90-6233-128-9

SISO 365.3 SVS 8.12.3 UDC 681.3.06

Trefw.: programmeren.

Uitgegeven door: Academic Service  
Postbus 96996  
2509 JJ 's-Gravenhage

Druk: Krips Repro Meppel

Bindwerk: Meeuwis, Amsterdam

Omslag ontwerp: JAM Gauw

Foto's omslag: Studio Olaf B. Boorsma

ISBN 90 6233 128 9

© 1984 A.S.; het auteursrecht berust bij de auteurs

Niets uit deze uitgave mag worden verveelvuldigd en/of openbaar gemaakt door middel van druk, fotocopie, microfilm, geluidsband, elektronisch of op welke andere wijze ook en evenmin in een retrieval system worden opgeslagen zonder voorafgaande schriftelijke toestemming van de uitgever.

---

## VOORWOORD

Deze text is in eerste instantie onder betrekkelijk hoge druk geschreven toen ons toch nog vrij plotseling een studierichting informatica werd opgedrongen. De reden om hem te schrijven was dezelfde als waarom wij er later in hebben toegestemd dat hij --in tweede versie-- in boekvorm wordt uitgegeven: de text voorziet in wat wij als ernstige behoefte zijn gaan beschouwen.

Programmeren is begonnen als een intuïtief bedreven ambacht. In 1968 brak de algemene erkenning door dat de tot dan gevolgde methoden van programmaontwikkeling ontoereikend waren om het hoofd te bieden aan wat zich toen als de zogenaamde "software crisis" manifesteerde. Nadat de toen alom gehanteerde operationele argumenten als essentieel ontoereikend waren geïdentificeerd, is de ontwerpstyl ontwikkeld waarin het programma en zijn correctheidsargument hand-in-hand worden ontworpen. Methodologisch was dat een dramatische vooruitgang.

Hoewel via monografie en leerboek algemeen toegankelijk, was deze verworvenheid nog niet tot het inleidend programmeeronderwijs doorgedrongen. Wat we dan maar als "de vlucht van de informatica" zullen aanduiden was om tweeeërlei reden aanleiding om hier een einde aan te maken. Ten eerste wordt het met toenemend studenten-aantal in toenemende mate onverantwoord het inleidend onderwijs op achterhaalde leest te schoeien. Ten tweede draagt de traditioneel intuïtieve inleiding met de toenemende popularisering van compu-

---

ters steeds minder bij tot de verdere vorming van de aankomende student.

Het is om die redenen dat in deze text programmeren gepresenteerd wordt als wat het inmiddels is geworden, namelijk een vrij formele tak van de wiskunde, waarin de mathematische logica een onmisbaar stuk gereedschap is geworden.

Het boek bestaat uit twee gedeelten, oorspronkelijk corresponderend met het college respectievelijk de bijbehorende instructie. Het college ontvouwt de stof die specifiek is voor het programmeren, de instructie beschrijft het daarbij gebruikte logische apparaat en bevat de oefeningen. Hoe de lezer zijn aandacht het beste over beide helften verdeelt wordt, omdat het optimum van zijn voorkennis afhangt, aan de lezer overgelaten.

Wij zijn dank verschuldigd aan alle collegae van de vakgroep informatica aan de THE die in de afgelopen jaren de hier beschreven stof met zoveel enthousiasme en succes hebben onderwezen. Expliciete vermelding verdienen A.J.M. van Gasteren, A. Kaldewaij, M. Rem, J.L.A. van de Snepscheut en J.T. Udding. Hun ervaring en stimulans zijn een grote steun geweest.

Eindhoven  
mei 1984

Edsger W. Dijkstra  
W.H.J. Feijen

DEEL 0



---

## EEN METHODE VAN PROGRAMMEREN

"Informatica" is de naam die in 1968 ten behoeve van de niet-Angelsaksische landen is bedacht voor het vak dat inmiddels in de Verenigde Staten van Amerika "computer science" en in Groot-Brittannië "computing science" heette. Voor de Angelsaksische term "computer" wordt in het Nederlands "automatische rekenmachine" of het kortere "rekenautomaat" gebezigd; beide termen zijn adequaat mits we --zoals we later zullen zien-- aan het begrip "rekenen" een niet te enge betekenis toekennen.

De term "automaat" bezigen we voor een mechaniek dat autonoom --d.w.z. zonder verder ingrijpen onzerzijds-- desgewenst iets voor ons kan doen. Een (althans in sommige landen) veel verbreide automaat is bijvoorbeeld de stortbak van een WC. Na het startsignaal --trekken aan de ketting of drukken op de knop-- verloopt de rest vanzelf: het toilet wordt schoongespoeld, de bak loopt vol en te rechter tijd wordt de toevoerkraan gesloten, zodat de bak niet overloopt.

Men zou op grond van het bovenstaande kunnen denken dat de sigarettenautomaat de naam "automaat" niet verdient, omdat de klant op allerlei wijzen moet ingrijpen: hij moet munten ingooien en een la opentrekken. Deze handelingen zijn evenwel te beschouwen als een uitgebreid startsignaal: het is een automaat voor de sigarettenboer, die tijdens de transactie er geen omkijken naar heeft.

Klassiekere voorbeelden van automaten zijn de klok en de speeldoos, die --mits opgewonden-- "O, mijn lieve Augustijn" speelt. (Het was vaak dezelfde ambachtsman die speeldozen en klokken --al dan niet van koekoek voorzien-- vervaardigde.)

Bovengenoemde automaten zijn een beetje saai omdat ze in zekere zin iedere keer hetzelfde doen: de stortbak verzorgt de ene doorspoeling na de andere, de klok herhaalt zijn patroon elke 12 uur en de speeldoos laat tot vervelens toe "O, mijn lieve Augustijn" horen. (Aangezien ook Watt's stoommachine tot deze groep van saaie automaten behoorde, betaamt het ons niet ons laatdunkend over die saaiheid uit te laten.)

Bovengenoemde automaten werden gevolgd door een flexibeler type, namelijk het type van de speeldoos met de verwisselbare rol: dat betekende dat met grotendeels hetzelfde apparaat "O, mijn lieve Augustijn", dan wel "Hop Marianneke, stroop in 't kanneke" ten gehore kon worden gebracht. Vele automaten zijn van dat type: de pianola, de filmprojector en het draaiorgel. Wederom betaamt het ons niet ons daar laatdunkend over uit te laten: het weefgetouw van Jacquard en de moderne band-gestuurde freesmachine vallen hier ook onder, evenals de afspeelapparatuur van grammofoonplaten, videoplatten en banden.

De bovengenoemde mechanieken zijn ten tonele gevoerd om het begrip "automaat" te illustreren. Voor het andere aspect van de rekenautomaat, namelijk dat hij "rekent", bieden ze echter geen aanknopingspunt, zodat we nu (onder dankzegging) weer afscheid van hen nemen.

Wat is rekenen? Laten we eens een heel eenvoudig voorbeeld bij de kop vatten: het in het tientallig stelsel optellen van twee natuurlijke getallen. Heel eenvoudig? Misschien zit dat nog:

nadat ze de cijfers 0 t/m 9 hebben geleerd, kost het kinderen op de lagere school nog jaren voordat ze de kunst onder de knie krijgen (en sommigen leren het nooit). Laat ons eens nagaan, wat daarbij komt kijken.

Om te beginnen leren we de tafels van optelling:

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

De bovenste rij en de linkerkolom zijn niet zo moeilijk en langzamerhand raakt het kind vertrouwd met de linkerbovenhoek van het tableau: het zogenaamde "rekenen onder de tien". Na enige tijd behoort ook de rechteronderhoek tot de parate kennis van het kind. Het kind kent nu uit het hoofd de som van twee getallen onder de tien. Dat is heel mooi: het kind kent nu het antwoord van 100 verschillende optellingen.

Maar het is ook duidelijk dat we zo niet door kunnen gaan: er zijn 10.000 verschillende optellingen van twee getallen onder de honderd, er zijn 1.000.000 verschillende optellingen van twee getallen onder de duizend, en het is kennelijk waanzin te proberen dergelijk grote tableaux uit je hoofd te kennen. Gelukkig hoeft dat ook niet, want --zoals het oplettende lezertje al zal hebben

opgemerkt-- het tableau is niet zonder regelmaat. Het volgende stadium van het optelonderwijs bestaat dan ook uit het leren exploiteren van deze regelmaat.

Het begint ermee, dat we grotere getallen niet als entiteit beschouwen, maar als rij van cijfers die we stuk voor stuk behandelen: uit de cijferrijen, die de addenda representeren, leren we de cijferrij construeren, die de som representeert.

Wat zijn de ingrediënten van dat constructieproces? Ten eerste worden de cijfers van de addenda twee-aan-twee aan elkaar toegevoegd, wat we weergeven door de getallen boven elkaar te schrijven. En we leren daarbij, dat  $2037 + 642$  er dan als

2037

642

en *niet* als

2037

642

hoort uit te zien.

Bovenstaande optelling is gemakkelijk omdat we paarsgewijs "onder de tien" kunnen rekenen:

2037

642 +

2679

en tot zover doet het er niet toe, of we van links naar rechts of van rechts naar links werken.

Om ook optellingen zoals

2037

645 +

2682

uit te kunnen voeren worden de regels uitgebreid met het "1 onthouden" en als klap op de vuurpijl wordt de leerling vertrouwd gemaakt met het cascadeverschijnsel dat optreedt wanneer de "onthouden 1" in rekening moet worden gebracht in een positie waar de som van de cijfers 9 is, zoals in

$$\begin{array}{r} 2057 \\ \underline{645} + \\ 2702 \end{array}$$

We zijn zo uitgebreid op de tientallige optelling van twee natuurlijke getallen ingegaan, niet omdat we veronderstellen dat de lezer niet kan optellen, maar om hem bewust te maken van de veelheid van regels die hij (met de nodige routine schier onbewust) toepast.

Mits voldoende scherp geformuleerd vormt zo'n samenstel van regels wat wij een *algorithm*e noemen. (Wij hebben boven een *algorithm*e voor de tientallige optelling van natuurlijke getallen losjes aangeduid.) Een *algorithm*e is een handelingsvoorschrift dat, mits getrouwelijk uitgevoerd, in een eindig aantal stappen tot het gewenste resultaat voert.

Naar aanleiding van de losjes aangeduide optelalgorithmen kunnen we meteen al de volgende opmerking maken.

Opmerking. Het is niet noodzakelijk zo, dat een *algorithm*e niets aan de fantasie van de uitvoerder overlaat: irrelevante keuzen mogen we openlaten. Om 2057 en 645 op te tellen, worden de getallen "boven elkaar" geschreven, maar kennelijk zijn

$$\begin{array}{r} 2057 \\ \underline{645} + \\ 2702 \end{array} \quad \text{en} \quad \begin{array}{r} 645 \\ \underline{2057} + \\ 2702 \end{array}$$

ons even lief. Bij de overeenkomstige vermenigvuldigingsalgorithm

is dit verschijnsel geprononceerder: vergelijk

71		28
<u>28</u>		<u>71</u>
568 *	en	28 *
<u>142</u>		<u>196</u>
1988 +		1988 +

(Einde van Opmerking.)

Opmerking. De optelalgorithme is in zoverre kenmerkend, dat hij in een zeer groot aantal verschillende gevallen kan worden toegepast. Dat, zoals in dit voorbeeld, de algorithme in een onbegrensd aantal gevallen toepasbaar is en er niet, onafhankelijk van de addenda, een bovengrens bestaat voor het aantal stappen dat een uitvoering van de algorithme vergen zal, neemt niet weg dat elke individuele uitvoering slechts een eindig aantal stappen vergt. (Einde van Opmerking.)

Een andere algorithme wordt gevormd door het agglomeraat van differentiatieregels dat ons bijvoorbeeld in staat stelt

$$\frac{d}{dx} (e^{\sin x})$$

te berekenen. Toegepast leiden zij in dit geval tot

$$(\cos x) \cdot e^{\sin x}$$

Ook de differentiatie-algorithme laat in het algemeen enige vrijheid --met name in de volgorde waarin de diverse regels worden toegepast-- , ook de differentiatie-algorithme is in principe in een onbegrensd aantal gevallen toepasbaar. Wij hebben dit voorbeeld opgenomen omdat, terwijl het gebruikelijk is te spreken over "de berekening van een afgeleide", het berekenen hier al ontdaan is van de heel specifieke numerieke associaties.

Andere voorbeelden van algorithmen zijn planimetrische constructies (voor de bisectrice van een hoek, het hoogtepunt van een driehoek, enzovoorts), breipatronen, gebruiksaanwijzingen, montagevoorschriften, recepten en de gedragsregels die wij volgen om te kijken of iemand voorkomt in het telefoonboek voor Amsterdam.

Opmerking. Bij het telefoonboek voor Amsterdam zijn de regels eenvoudiger en vergt het zoeken gemiddeld veel minder stappen dan bij het telefoonboek voor Eindhoven en Omstreken anno 1982, waarin de namen van de aangeslotenen dorpsgewijze zijn gerangschikt. Ten aanzien van het laatste telefoonboek zou men dan ook van een ontwerpfout kunnen spreken. (Einde van Opmerking.)

De automatische rekenmachine heet zo omdat hij automatisch kan "rekenen" in de zin van: automatisch een algoritme uitvoeren. De rekenautomaat ontleent zijn grote flexibiliteit aan het feit dat de keuze van de algoritme die de automaat zal uitvoeren aan ons is en dat we bij de keuze van die algoritme een schier onbegrensde vrijheid hebben. (Vergeleken bij de eerder genoemde automaten representeert de rekenautomaat dan ook een "quantum jump".)

Dat de rekenautomaat gevoed kan worden met een algoritme naar keuze wordt uitgedrukt door te zeggen dat de rekenautomaat "programmeerbaar" is. Een algoritme die door een automaat uitgevoerd zou kunnen worden heet een "programma" --letterlijk "voorschrift"-- en het ontwerpen van programma's heet "programmeren". Programmeren is het hoofdonderwerp van dit college.

Om een aantal redenen is programmeren een college waard. Ten eerste omdat er iedere keer een programma nodig is om de brug te slaan tussen de "general purpose computer" en de specifieke toepassing en daardoor de activiteit van het programmeren een centrale plaats inneemt. Ten tweede omdat wij uit ervaring weten dat, wie niet geleerd heeft tijdens het programmeren voldoende zakelijk en

betrouwbaar over zijn ontwerp te denken en te redeneren, onherroepelijk brokken maakt. De student volledig vertrouwd maken met de meest effectieve manier van redeneren over algoritmen, die bekend is, is dan ook een belangrijke doelstelling van dit college.

Eén waarschuwing is op zijn plaats: een programma is een formele text waarin iedere letter, elk cijfer, elk interpunctieteken en elke operator zijn rol speelt. Programma's moeten daardoor met ongewone nauwkeurigheid worden opgesteld. Omdat de meeste mensen zijn opgegroeid met het idee dat ze zich in hun geschriften hier en daar best wat schrijf- en spelfouten mogen permitteren en gewend zijn zich dienovereenkomstig te gedragen, schrikken mensen bij hun eerste kennismaking met programmeren vaak zó van deze eis van zorgvuldigheid, dat ze denken dat programmeren alleen maar een kwestie van accuratesse is. Wanneer deze accuratesse een tweede natuur is geworden, realiseert men zich dat de ware moeilijkheid heel ergens anders ligt: die ligt in de plicht te voorkomen dat het ontwerp onhanteerbaar ingewikkeld wordt. (Heel onervarenen vertalen de noodzaak van deze accuratesse in een verwijt aan de rekenmachine, maar zij realiseren zich niet dat de rekenautomaat zijn bruikbaarheid nu juist ontleent aan de getrouwheid waarmee hij de hem toegevoegde algoritme *en geen andere* uitvoert.)

Tenslotte verzoeken wij de student zich te realiseren dat wat in het beperkte bestek van dit inleidend college behandeld kan worden, niet representatief kan zijn voor programmeren in al zijn mogelijke facetten. Om tijd te besparen en het niet moeilijker te maken dan nodig is zullen wij onze programma's ontwikkelen voor een heel sobere machine waaraan vele toeters en bellen (die maar al te vaak haken en ogen blijken) ontbreken. Aan de specifieke moeilijkheden van het ontwikkelen van werkelijk grote programma's zullen wij in dit inleidend college niet toekomen.

## Automaten en hun toestanden

Laat ons een automaat beschouwen die, indien gestart, een tijdje iets doet en dan stopt. Als we voorbeelden in gedachten willen hebben kunnen we denken aan een grammofoon of een stortbak. De grammofoon wordt gestart door een plaat op te leggen en de naald in de inloopgroef te laten zakken; hij stopt als de naald via de uitloopgroef dicht genoeg bij de as van de draaitafel komt. De stortbak wordt gestart door aan de ketting te trekken; als de bak weer vol is gelopen, wordt de toevoerkraan afgesloten en stopt het proces.

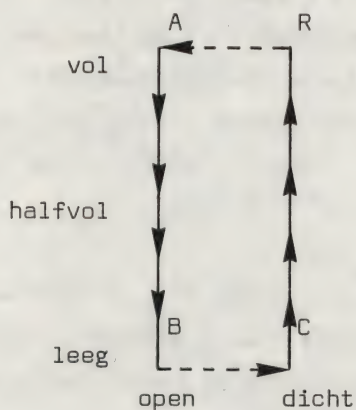
Niet alleen dat een automaat, indien gestart, een tijdje iets doet en dan stopt, zijnde een automaat doet hij dat autonoom, d.w.z. zonder verder ingrijpen onzerzijds. Een gevolg hiervan is dat een dergelijke automaat zich op ieder moment tussen start en stop in een *verschillende* toestand bevindt: kort na de start bevindt hij zich in een toestand zodat hij nog een heel tijdje werkt, vlak voor het einde bevindt hij zich in een toestand zodat hij bijna stopt.

Wie de automaat in kwestie kent en weet waarnaar hij kijken moet, kan dan ook altijd zien hoever de werkende automaat is gevorderd. In het geval van de grammofoon wordt de staat van vordering bijvoorbeeld weerspiegeld in de positie van de arm: één blik op de positie van de arm is voldoende om vast te stellen, hoever het afspelen van de plaat is gevorderd.

Opmerking. Wij vestigen er de aandacht op dat op verschillende momenten tussen start en stop de automaat zich in verschillende toestanden moet bevinden. Wanneer door een kras in de plaat de naald terugspringt in de vorige groef en de grammofoon daarmee terugkeert in een toestand waarin hij zich sinds de start al heeft

bevonden, is aan deze voorwaarde niet voldaan. Er is dan ook iets mis: hij "blijft in de groef hangen" en zal niet autonoom stoppen. (Einde van Opmerking.)

Ook in het geval van de stortbak legt de toestand vast, hoever het autonome proces is gevorderd. Het waterpeil in de bak neemt de rol van de positie van de arm evenwel slechts gedeeltelijk over: gedurende de hele cyclus is de bak immers twee keer halfvol, één keer tijdens het leeglopen en één keer tijdens het vollopen. Het onderscheid daartussen wordt bepaald door het feit of de klok de afvoer niet of wel afsluit. We komen tot de conclusie dat de toestand van de stortbak --in benadering-- wordt vastgelegd door twee variabelen: de continue variabele "waterhoogte" en de discrete variabele "afvoer", waarvoor slechts de twee waarden "open" en "dicht" ter beschikking staan.



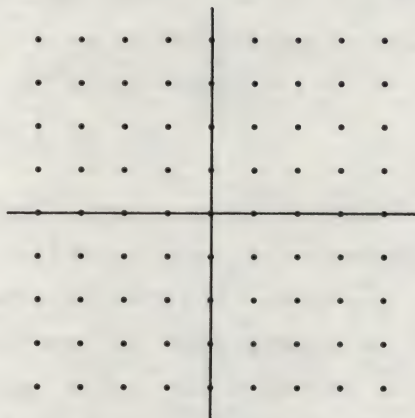
In bovenstaand plaatje hebben we de waterhoogte verticaal uitgezet en de twee standen van de afvoer, "open" en "dicht", in de horizontale richting. Het punt R is de rusttoestand: bak vol en afvoer dicht. De start --het trekken aan de ketting-- opent de afvoer, die open blijft zolang water er met voldoende snelheid doorheen stroomt. Als de bak leeg raakt, zakt de klok weer en

wordt de afvoer gesloten, waarna de bak volloopt. (De watertoevoer is geopend wanneer de bak niet vol is. De capaciteit van de toevoer is echter kleiner dan die van de afvoer, zodat de toevoer de bak niet belet leeg te raken. Verifieer dat, wanneer de capaciteit van de afvoer twee keer zo groot is als die van de toevoer, het doorspoelen van het toilet --d.w.z. het doorlopen van het pad van A naar B -- even lang duurt als het daarna vullen van de bak --d.w.z. het doorlopen van het pad van C naar R -- . Uit het feit dat in het algemeen het doorspoelen veel korter duurt dan het daarna weer vollopen van de bak, kunnen we concluderen dat gemeenschappelijk de verhouding van de twee capaciteiten aanmerkelijk groter is dan 2 .)

Elke mogelijke toestand van de stortbak komt overeen met een punt in ons twee-dimensionale plaatje, dat daarom de naam "toestandsruimte" heeft gekregen. (In dit speciale geval zouden we van "toestandsvlak" kunnen spreken omdat de toestandsruimte slechts twee-dimensionaal is. Wij doen dit niet en gebruiken de meer algemene term "ruimte"; in vele gevallen zal onze toestandsruimte dan ook meer dan twee dimensies hebben.) De geschiedenis die zich afspeelt in de periode van start tot stop vindt zijn weerspiegeling in de vorm van een pad in de toestandsruimte dat doorlopen wordt. (Merk op dat het pad niet weerspiegelt met welke snelheid het doorlopen wordt.)

De positie van een punt in de toestandsruimte is hier gegeven door twee coördinaten: de waterhoogte en het feit of de afvoer open of dicht is. De waterhoogte is hier een continue variabele, de toestand van de afvoer is hier behandeld als een discrete variabele, die alleen de waarden "open" en "dicht" kent. (Dat wil zeggen, dat wij openen en sluiten van de afvoer als ondeelbare, "instantane" gebeurtenissen beschouwen: de afvoer is open of dicht, maar niet "half open". Dit soort "puntgebeurtenissen" is

een bruikbare idealisering, niet ongelijk aan de "puntmassa" in de klassieke mechanica.) Met het oog op de structuur van rekenautomaten zullen wij ons in het vervolg beperken tot toestandsruimten waarin alle coördinaten discreet zijn. Als bijvoorbeeld de toestandsruimte wordt opgespannen door twee integer coördinaten --d.w.z. beperkt tot geheeltallige waarden-- , dan kunnen we de toestandsruimte voorstellen door de roosterpunten in het platte vlak



en het pad door "sprongetjes" van het ene roosterpunt naar het andere.

Opmerking. Er bestaan machines, waarin waarden gerepresenteerd worden door continu variabele fysische grootheden, zoals voltage, stroomsterkte of rotatiehoek van een as. Dit zijn zogenaamde "analogon machines". Zij vallen geheel buiten het bestek van dit college. Vanouds hebben analogon machines het nadeel gehad dat het technisch onmogelijk is op deze wijze waarden met erg grote precisie te representeren. Hun vroegere voordeel van snelheid hebben ze met het steeds sneller worden van discrete rekenapparatuur grotendeels verloren, waardoor steeds meer van wat vroeger door analogon apparatuur gedaan werd nu met discrete apparatuur geschiedt - denk bijvoorbeeld aan "digital recording" van muziek. (Einde van Opmerking.)

## De berekening als toestandsverandering

Een rekenautomaat die gebruikt wordt reageert op signalen die hij van de buitenwereld ontvangt en doet zulks door in respons op de ontvangen signalen op zijn beurt weer signalen af te geven. Het opnemen van informatie uit de buitenwereld heet de "invoer" (= input), het afgeven van informatie heet "uitvoer" (= output). Als onderdeel van de automatisering van 008 --het nummer voor "Inlichtingen" van de telefoondienst-- kunnen we ons een machine voorstellen, die als invoer naam en adres van een telefoonabonnee opneemt en als uitvoer het bijbehorende telefoonnummer aan de buitenwereld retourneert.

De manier waarop invoer en uitvoer plaatsvinden pleegt van automaat tot automaat te verschillen. Bij de automatisering van 008 zal de invoer van naam en adres waarschijnlijk plaatsvinden doordat de telefoniste deze via een toetsenbord (als van een schrijfmachine) "intikt", terwijl de uitvoer via een beeldscherm plaatsvindt, zodat de telefoniste zonder bladeren in grote telefoonboeken het gevraagde antwoord kan geven. Het is ook denkbaar dat de automaat het antwoord in verstaanbare vorm produceert. De girodienst levert ons een voorbeeld waar in- en uitvoer via heel andere kanalen plaatsvinden: invoer vindt plaats via ponskaarten (of andere mechanisch leesbare formulieren) en uitvoer vindt plaats met behulp van geadresseerde en verder bedrukte "berichten van overschrijving".

Vanwege de grote variëteit van invoer- en uitvoermedia zullen wij in dit college van invoer en uitvoer grotendeels abstraheren en onze aandacht concentreren op wat zich afspeelt tussen het moment waarop de invoer voltooid is en het rekenwerk dus kan begin-

nen en het moment waarop het rekenwerk voltooid is en het antwoord dus voor uitvoer gereed ligt.

Opmerking. Eenvoudigheidshalve gaan wij er aan voorbij dat soms het rekenproces gedeeltelijk al kan beginnen voordat de invoer voltooid is en dat soms de uitvoer van een gedeelte van het antwoord al mogelijk is voordat het rekenproces helemaal voltooid is. (Einde van Opmerking.)

De redenen om ons te beperken tot de periode van het einde van de invoer tot het begin van de uitvoer zijn velerlei. Ten eerste lijken verschillende rekenautomaten in wat zich dan afspeelt veel meer op elkaar dan in de manieren waarop zij met de buitenwereld communiceren; wat zich dan afspeelt is daardoor een onderwerp van veel algemenere geldigheid. Ten tweede is het gedurende deze periode dat het eigenlijke rekenproces zich afspeelt, waarop wij onze aandacht dienen te concentreren. (Een derde reden kunnen wij nu slechts noemen, maar nog niet uitleggen: het is een vereenvoudiging die ons in staat stelt deelberekeningen op dezelfde voet te behandelen als de totale berekening.)

In het volgende zullen wij rekenprocessen beschouwen die uitgaan van een *begintoestand* van de automaat en leiden tot een *eindtoestand* van de automaat. Indien het de gehele berekening betreft zullen wij in het vervolg stilzwijgend aannemen dat de begin-toestand rechtstreeks door de invoer is bepaald en dat de eindtoestand rechtstreeks bepaalt wat uitgevoerd dient te worden.

Iets precieser: voor elke berekening worden begin- en eindtoestand beschreven door dezelfde collectie coördinaten, de invoer bestaat daaruit dat voor de begintoestand de waarde van één of meer coördinaten wordt voorgeschreven en in de eindtoestand representeert de waarde van één of meer coördinaten het gewenste antwoord.

Opmerking. Het is niet noodzakelijk dat de invoer de waarde van alle coördinaten voorschrijft. (Einde van Opmerking.)

\*       \*       \*

Nu wij besloten hebben de berekeningen te beschouwen als toestandsveranderingen kunnen wij de zogenaamde *functionele specificatie van een programma* geven door op te geven hoe begin- en eindtoestand dienen samen te hangen. Voor deze functionele specificaties zullen wij een heel vast schema volgen, dat we nu zullen beschrijven en met een serie kleine voorbeeldjes zullen illustreren.

Zo'n functionele beschrijving bestaat uit 4 ingrediënten, in volgorde

- (i) de zogenaamde *declaratie* van de zogenaamde *lokale variabelen*
- (ii) de beginconditie --traditioneel tussen accolades geplaatst--
- (iii) de naam van het programma --traditioneel van het voorafgaande gescheiden door een puntkomma--
- (iv) de eindconditie --traditioneel eveneens tussen accolades geplaatst-- .

Het geheel wordt voorafgegaan door de openingshaak "[[" --uit te spreken als "begin"-- en gevolgd door de bijbehorende sluitingshaak "]]" --uit te spreken als "end"-- .

Een heel eenvoudig voorbeeld van een functionele specificatie is de volgende --om der wille van de discussie hebben we de regels overeenkomstig gemarkeerd--

```
(i)      |[ x: int
(ii)      {x = X}
(iii)     ; skip
(iv)      {x = X}
          ]| .
```

Opmerking. Als openings- en sluitingshaak niet op dezelfde regel staan worden zij om der wille van de overzichtelijkheid recht onder elkaar geplaatst. De voorafgaande functionele specificatie is zo klein dat er helemaal geen bezwaar zou zijn geweest tegen de layout:

```
[[ x: int {x = X}; skip {x = X} ]]
```

(Einde van Opmerking.)

Wij moeten bovenstaande functionele specificatie als volgt lezen. Regel (i) vertelt ons drie dingen: dat het hier een toestandsruimte met slechts één coördinaat betreft, dat deze met de naam "x" wordt aangeduid en dat zijn waardebereik beperkt is tot de gehele getallen. Dit laatste is de portée van de zogenaamde *typeaanduiding* ": int", waarmee de declaratie is afgesloten. ("int" is de afkorting van het Latijnse woord "integer" dat in het Engels voor de gehele getallen gebruikt wordt.) De rest vertelt ons dat de aanvankelijke geldigheid van de beginconditie (ii) voldoende is opdat uitvoering van het programma "skip", zoals dit in regel (iii) benoemd is, ervoor zorgt dat na afloop de eindconditie (iv) geldt.

Wanneer er, zoals hier het geval is, in de condities een grootheid staat zoals  $X$ , die eigenlijk zomaar uit de lucht komt vallen, dan betekent dat dat de functionele specificatie geldt voor elke *mogelijke* waarde van  $X$ . (Omdat  $x$  geheel is en  $x = X$  moet gelden, hoeven we ons om  $X = 3\frac{1}{2}$  niet te bekommeren.) M.a.w. de bovenstaande functionele specificatie van "skip" vertelt ons dat "skip" de waarde van  $x$ , wat die ook zijn moge, ongewijzigd dient te laten.

Een nauwelijks ambitieuzer voorbeeld van een functionele specificatie van een programma, dat we gemakshalve ook maar weer "skip" noemen, is

```
(i)      |[ x, y, z: int
(ii)      {x = X  ^  y = Y  ^  z = Z}
(iii)     ; skip
(iv)      {x = X  ^  y = Y  ^  z = Z}
      ]|
```

Het bovenstaande schrijft voor dat in een drie-dimensionale toestandsruimte met coördinaten  $x$ ,  $y$  en  $z$  uitvoering van "skip" de waarde van elk der coördinaten, wat die ook zijn moge, ongewijzigd dient te laten.

Opmerking. De volgorde waarin, onderling door komma's gescheiden, de lokale namen  $x$ ,  $y$  en  $z$  in de declaratie worden opgesomd, doet niet ter zake. Equivalente vormen voor regel (i) zijn derhalve:

```
(i)      |[ x, z, y: int
(i)      |[ z, y, x: int
```

enzovoorts.

Wij hebben hier de drie lokale variabelen gezamenlijk in één declaratie geïntroduceerd. We hadden hen ook elk met hun eigen declaratie mogen invoeren; zulke onafhankelijke declaraties moeten dan wel onderling door een puntkomma gescheiden zijn. Bijvoorbeeld

```
(i)      |[ x: int; y: int; z: int
```

Ook mengvormen zijn toegestaan, zoals in

```
(i)      |[ x, y: int; z: int
```

Opgave. Ga na, dat met bovenstaande vrijheden regel (i) in 24 verschillende doch equivalente vormen kan worden opgeschreven. (Einde van Opgave.)

Als in een ingewikkeld programma variabelen duidelijk groeps-  
gewijze bij elkaar horen, komt een overeenkomstig groeps-  
gewijze

declaratie de overzichtelijkheid van de tekst soms ten goede.  
(Einde van Opmerking.)

Vooraan de declaratie(s) staat altijd de openingshaak "[[" . De bijbehorende sluitingshaak "]]" geeft aan tot welk punt van de tekst de betekenis van toepassing is die door de declaratie aan de geïntroduceerde namen is toegekend. Op deze wijze begrenst het hakenpaar "[[" en "]]" in de tekst de reikwijdte (= scope) van de naamgeving.

Wij hebben hierboven twee specificaties gegeven, eentje voor "skip" in een één-dimensionale, en eentje voor "skip" in een drie-dimensionale toestandruimte. Op deze wijze zouden wij "skip" even vaak kunnen definiëren als wij toestandruimten introduceren. Omdat dat veel te eentonig zou worden achten wij "skip" in elke toestandruimte gedefinieerd: "skip" is in het vervolg de universele naam van de handeling die niets aan de toestand verandert, ongeacht welke toestandruimte voor het vastleggen van de toestand gebruikt wordt.

Opmerking. Op het oog lijkt "skip" niet zo vreselijk nuttig. Wij zullen later zien, dat "skip" ongeveer net zo nuttig is als het cijfer 0 in het positietalstelsel. (Einde van Opmerking.)

Verdere voorbeelden van functionele specificaties zijn

```
[[ x, y: int
  {x = X  ^  y = Y}
; swap0
  {x = Y  ^  y = X}
]]
```

en

```
[[ x, y: int
  {x = Y  ∧  y = X}
; swap1
  {x = X  ∧  y = Y}
]]
```

Opgave. Toon aan dat de functionele specificaties van swap0 respectievelijk swap1 equivalent zijn. (Einde van Opgave.)

Programma's swap0 en swap1 hebben de eigenschap dat uit de kennis van de eindtoestand afgeleid kan worden wat de begintoestand is geweest. Als bijvoorbeeld na afloop van de uitvoering van swap1 de toestand door  $x = 2 \wedge y = 3$  gegeven is, concluderen we dat in de begintoestand  $x = 3 \wedge y = 2$  heeft gegolden. We drukken dit uit door te zeggen dat de uitvoering van swap1 geen informatie vernietigt, zulks in tegenstelling tot bijvoorbeeld de volgende programma's copy en euclid.

```
[[ x, y: int
  {x = X}
; copy
  {x = X  ∧  y = X}
]]
```

Als copy eindigt met  $x = 5 \wedge y = 5$  kunnen we wel concluderen dat aanvankelijk  $x = 5$  heeft gegolden. Over de aanvangswaarde van y --die immers *niet* in de beginconditie genoemd wordt-- kunnen we niets concluderen; de waarde van y kan aan het begin dus alles geweest zijn.

```
[[ x, y: int
  {x = X  ∧  y = Y  ∧  x > 0  ∧  y > 0}
; euclid
  {x = y  ∧  x = gcd(X, Y)}
]]
```

waarin "ggd" staat voor "grootste gemene deler van". Als euclid eindigt met  $x = 5 \wedge y = 5$ , kan de begintoestand niet alles zijn geweest --  $x = 40 \wedge y = 70$  is bijvoorbeeld uitgesloten-- maar er zijn wel vele mogelijkheden. We hadden hetzelfde programma euclid ook als volgt mogen specificeren:

```

|| x, y: int
    {X = ggd(x, y)  $\wedge$  x > 0  $\wedge$  y > 0}
; euclid
    {x = X  $\wedge$  y = X}
|| .

```

Inspectie van de eindtoestand legt wel de waarde van X vast --bijvoorbeeld 5 -- maar voor gegeven waarde van X heeft de beginconditie, gezien als vergelijking in de twee onbekenden x en y, vele oplossingen. Ook euclid vernietigt dus informatie.

Opmerking. Men realiseere zich dat een onjuiste functionele specificatie het onmogelijke kan vergen, zoals in de volgende onjuiste specificatie voor root :

```

|| x: int
    {x = X}
; root
    {x =  $\sqrt{X}$ }
|| .

```

Als aanvankelijk  $x = 43$ , dan is voor  $X = 43$  netjes aan de beginconditie voldaan maar met die waarde van X kan de eindconditie met gehele (!) x niet worden bevredigd! (En soortgelijk wanneer aanvankelijk x negatief zou zijn.) Kortom: de beginconditie als boven is te zwak en moet versterkt worden met de neveneis dat x een kwadraat is. Indien --zoals hier en meestal-- de eindtoestand uniek door de begintoestand bepaald is, is het benoemen van de waarden in de eindtoestand als regel de makkelijkste manier om dit te ondervangen:

```
l[ x: int
    {x = X2  ∧  X ≥ 0}
; root
    {x = X}
]| ... (Einde van Opmerking.)
```

## Programma's en hun opbouw

In het voorafgaande hebben wij gezien dat het de bedoeling van een berekening is een toestandsverandering te bewerkstelligen zoals die in een functionele specificatie is vastgelegd. Doordat zo'n functionele specificatie als regel een aantal ongespecificeerde parameters bevat --in onze voorbeelden met  $X$ ,  $Y$  of  $Z$  aangeduid-- beschrijft één functionele specificatie als regel een grote klasse van toestandsveranderingen. Het programma dient aan te geven hoe deze toestandsveranderingen bewerkstelligd dienen te worden.

Voor een beperkte klasse toestandsveranderingen kan dit in één enkele stap: de begintoestand wordt dan direct in de eindtoestand overgevoerd. Zulke berekeningen duren heel kort; zij corresponderen met de bouwstenen waaruit wij gecompliceerde programma's kunnen opbouwen, die tot langere berekeningen aanleiding kunnen geven. Gedurende zo'n langere berekening wordt de totale toestandsverandering bewerkstelligd door de opeenvolging van een (eventueel groot) aantal "kleine" veranderinkjes, dat wil zeggen die veranderingen die in één stap direct kunnen worden geëffectueerd. De overeenkomstige bouwstenen waaruit het programma is opgebouwd heten "assignment statements" en de volgende sectie behandelt hoe assignment statements in een programma worden genoteerd en welke toestandsveranderingen met elke assignment statement overeenkomen.

Terzijde. Wij maken tweeledig gebruik van ons formalisme voor functionele specificatie. Bij euclid was de functionele specificatie gedacht als de formulering van een programmeeropgave. Bij skip werd de functionele specificatie gebruikt ter definitie van een bouwsteen die de programmeur ter beschikking staat. Dit laatste gebruik zullen wij ook volgen bij de introductie van de assignment statement. (Einde van Terzijde.)

### De assignment statement

Beschouw de functionele specificaties van de volgende programma's --die we uit gebrek aan fantasie  $S_0, S_1, S_2, \dots$  zullen noemen-- .

```
[ [ x, y: int {X = 0  ∧  Y = y}; S0 {X = x  ∧  Y = y} ] |
[ [ x, y: int {X = 88  ∧  Y = y}; S1 {X = x  ∧  Y = y} ] |
[ [ x, y: int {X = x + 3  ∧  Y = y}; S2 {X = x  ∧  Y = y} ] |
[ [ x, y: int {X = 7 * y  ∧  Y = y}; S3 {X = x  ∧  Y = y} ] |
[ [ x, y: int {X = 10 *(x - y)  ∧  Y = y}; S4 {X = x  ∧  Y = y} ] |
```

Opmerking. Het was op zichzelf eenvoudiger geweest om voor  $S_0$  te specificeren

```
[ [ x, y: int {Y = y}; S0 {x = 0  ∧  y = Y} ] | .
```

Ook deze specificatie drukt netjes uit dat na afloop van de uitvoering van  $S_0$  de waarde van  $x$  (ongeacht zijn aanvangswaarde:  $x$  komt in de beginconditie immers niet voor!) = 0 is en de waarde van  $y$  onveranderd is gebleven. (Ga na, dat de functionele specificatie van  $S_1$  op soortgelijke wijze vereenvoudigd zou kunnen worden.) Wij hebben in het bovenstaande lijstje deze vereenvoudiging met opzet niet doorgevoerd om de functionele

specificaties zoveel mogelijk te laten overeenstemmen. (Einde van Opmerking.)

Boven gegeven functionele specificaties volgen allemaal hetzelfde patroon, namelijk

$$I[ x, y: \text{int} \{X = E \wedge Y = y\}; S_i \{X = x \wedge Y = y\} ]I$$

met voor E respectievelijk 0, 88,  $x + 3$ ,  $7 * y$ ,  $10 * (x - y)$  --en met voor  $S_i$  de gekozen naam--.

Onze programma-notatie biedt de mogelijkheid zulke programma's te noteren, en wel als volgt met behulp van zogenaamde "assignment statements"

```

voor S0:      x:= 0
voor S1:      x:= 88
voor S2:      x:= x + 3
voor S3:      x:= 7 * y
voor S4:      x:= 10 *(x - y)

```

(Spreek de zogenaamde assignment operator "x:=" uit als "wordt", dus "x wordt nul", "x wordt achtentachtig", "x wordt x plus drie", "x wordt zeven maal y" en "x wordt tien maal haakje open x min y haakje dicht".)

Het postulaat van de assignment behelst dat voor elke toelaatbare expressie E het programma "x:= E" de functionele specificatie

$$I[ x, y: \text{int} \{X = E \wedge Y = y\}; x := E \{X = x \wedge Y = y\} ]I$$

bevredigt. Voor x mag elke gedeclareerde variabele worden gekozen; het postulaat van de assignment behelst dus evenzeer dat voor elke toelaatbare expressie E het programma "y:= E" de functionele specificatie

$$I[ x, y: \text{int} \{X = x \wedge Y = E\}; y := E \{X = x \wedge Y = y\} ]I$$

kunnen we  $x$ ,  $y$  en  $z$  elimineren door dat beginstuk te laten vervallen. We vinden dan

```

| [ x, y, z: int
    {  $x + 3 \geq y$  }
    ;  $x := x + 3$ 
    {  $x \geq y$  }
| | .

```

In woorden: de aanvankelijke geldigheid van  $x + 3 \geq y$  veroorlooft de conclusie dat na afloop van de uitvoering van de assignment statement  $x := x + 3$  de betrekking  $x \geq y$  geldt. (Deze conclusie is duidelijk zwakker dan wat we al wisten: dat de uitvoering van  $x := x + 3$  de waarden van  $y$  en  $z$  onverlet laat, wordt er niet meer in uitgedrukt.)

In het voorafgaande was  $x + 3$  een heel speciale keuze voor de rechterkant van de assignment statement. Analooch hadden we voor elke toelaatbare expressie  $E$  uit het postulaat van de assignment kunnen afleiden

```

| [ x, y, z: int {  $E \geq y$  };  $x := E$  {  $x \geq y$  } | | .

```

Ook de keuze van de eindconditie  $x \geq y$  was willekeurig. Hadden wij als eindconditie bijvoorbeeld  $z \cdot (x + 1) \leq (y + 3) \cdot x$  gekozen, dan hadden wij afgeleid

```

| [ x, y, z: int
    {  $z \cdot (E + 1) \leq (y + 3) \cdot E$  }
    ;  $x := E$ 
    {  $z \cdot (x + 1) \leq (y + 3) \cdot x$  }
| | .

```

Het algemene patroon om voor de assignment statement  $x := E$  en gegeven eindconditie  $R$  de bijbehorende beginconditie te vinden

is blijkbaar dat we in  $R$  voor  $x$  de expressie  $E$  --zo nodig tussen haakjes-- substitueren. Het is gebruikelijk dit substitutieresultaat met  $R_E^x$  aan te duiden. Met die conventie kunnen we onze regel samenvatten als

$$|[x, y, z: \text{int } \{R_E^x\}; x := E \{R\}]| \quad .$$

Opmerking. Toepassing van de regel op de eindconditie  $\neg R$  levert ons de uitspraak

$$|[x, y, z: \text{int } \{\neg R_E^x\}; x := E \{\neg R\}]| \quad .$$

We zien hieruit dat de aanvankelijke geldigheid van  $R_E^x$  niet alleen voldoende, maar ook nodig is opdat  $x := E$  een toestand bewerkstelligt waarin  $R$  geldt. (Einde van Opmerking.)

## Toelaatbare expressies

Een programma is een handelingsvoorschrift dat door een automatische rekenmachine moet kunnen worden uitgevoerd. Dat betekent dat er geen misverstand over mag bestaan wat dat handelingsvoorschrift behelst. Weinig mensen zullen na het voorafgaande er aan twifelen dat met de assignment statement

$$x := 2 * x$$

bedoeld wordt dat zijn uitvoering de waarde van  $x$  verdubbelt. Maar over de bedoeling van

$$x := x / 2 * 6$$

zijn (als je het maar aan genoeg mensen vraagt) de meningen verdeeld. In Nederland --waar volgens de traditie vermenigvuldiging voor deling gaat-- zal de opvatting dat

$$x := x / (2 * 6)$$

bedoeld wordt wel opgeld doen. In landen met andere tradities wordt evenwel de betekenis

$$x := (x / 2) * 6$$

verondersteld. Het is duidelijk dat dit soort dubbelzinnigheden door scherpe definities uitgebannen dient te worden.

Het uitbannen van dit soort dubbelzinnigheden brengt onvermijdelijk met zich mee dat even scherp gedefinieerd wordt van welke expressies de betekenis ondubbelzinnig vastligt. Het is deze definitie, waarmee dit hoofdstuk zich bezighoudt. En passant zullen wij het wijdstverbreid formalisme introduceren dat voor het geven van dergelijke definities in gebruik is, namelijk BNF (= Backus-Naur-Form, zo genoemd naar John Backus en Peter Naur; BNF heeft zijn bekendheid gekregen door de wijze waarop het gebruikt is in het beroemde "ALGOL 60 Report" van jan. 1960.) .

Zowat de eenvoudigste toelaatbare expressie is het natuurlijke getal. Wij zullen nu BNF gebruiken om te definiëren hoe natuurlijke getallen er op papier uitzien. En omdat de notatie voor natuurlijke getallen uit cijfers is opgebouwd, zullen we eerst definiëren welke verschijningsvormen er voor cijfers zijn. In BNF wordt dit gegeven door de zogenaamde "syntaxregel"

< cijfer > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 .

Links van het teken "::=" --uit te spreken als "is gedefinieerd als"-- staat de naam van de syntactische eenheid die gedefinieerd wordt, geplaatst tussen hoekige haakjes, rechts van het teken "::=" staan de verschijningsvormen van de syntactische eenheid, onderling gescheiden door een verticaal streepje "|" --uit te spreken als "of"-- . De regel vertelt ons, welke 10 karakters cijfers zijn en, bovendien, dat als wij later in een syntactische

formule `< cijfer >` tegenkomen, dit voor elk van deze 10 karakters kan staan.

Opmerking. De volgorde, waarin de alternatieve verschijningsvormen in een syntaxregel worden opgesomd, doet niet ter zake. We hadden de syntactische eenheid `cijfer` evengoed door

`< cijfer > ::= 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0`

kunnen definiëren. (Einde van Opmerking.)

Nu hebben we het gereedschap om --als we dat zouden willen-- te definiëren welke karakterrijen behoren tot de syntactische eenheid "getal onder de duizend":

`< getal onder de duizend >`

`::= < cijfer >`

`| < cijfer > < cijfer >`

`| < cijfer > < cijfer > < cijfer > .`

Bovenstaande definitie vertelt ons, dat een getal onder de duizend drie alternatieve verschijningsvormen heeft: een enkel cijfer, twee cijfers achter elkaar of een rijtje van drie cijfers. Om op deze manier de verschijningsvormen van "getal onder de milliárd" op te schrijven wordt een nare schrijfles, om op deze manier de verschijningsvormen van een natuurlijk getal op te schrijven is onbegonnen werk. In BNF wordt de syntactische eenheid natuurlijk getal gegeven door

`< natuurlijk getal >`

`::= < cijfer >`

`| < cijfer > < natuurlijk getal > .`

Dit is een zogenaamde "recursieve definitie": de syntactische eenheid die gedefinieerd wordt, komt voor in zijn eigen definitie (namelijk in het 2de alternatief)! De eerste confrontatie met een

recursieve definitie roept gewoonlijk enige huivering op: men moet onwillekeurig denken aan de slang die zich bij zijn eigen staart begint op te eten totdat er niets meer over is. Nadat de eerste huivering is overwonnen leert men evenwel de recursieve definitie waarderen: zonder dat zouden wij namelijk niet in staat zijn een syntactische eenheid met een onbegrensd aantal verschijningsvormen te definiëren.

Als we er even over nadenken is bovenstaande definitie van (de syntax van) natuurlijk getal dankzij de aanwezigheid van het 1ste alternatief niet zo griezelig als hij er op het eerste gezicht misschien uitziet. Dankzij de definitie van cijfer geeft het 1ste alternatief ons 10 verschijningsvormen van een natuurlijk getal (en daarmee is het 1ste alternatief als het ware "uitgeput"). Met deze 10 verschijningsvormen als mogelijk substituuut voor  $\langle \text{natuurlijk getal} \rangle$  in het 2de alternatief levert dat 100 nieuwe verschijningsvormen, met die 100 als mogelijk substituuut in het 2de alternatief krijgen we 1000 nieuwe verschijningsvormen, enzovoorts, enzovoorts.

Opmerking. De syntaxregel

$$\begin{aligned} \langle \text{natuurlijk getal} \rangle & \\ ::= \langle \text{cijfer} \rangle & \\ | \langle \text{natuurlijk getal} \rangle \langle \text{cijfer} \rangle & \end{aligned}$$

is equivalent met de eerder gegevene. Beide definiëren de verzameling eindige, niet-lege rijen cijfers. (Einde van Opmerking.)

Analoog aan onze definitie van cijfer definiëren we

$$\begin{aligned} \langle \text{letter} \rangle ::= & a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid \\ & n \mid o \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z \mid \\ & A \mid B \mid C \mid D \mid E \mid F \mid G \mid H \mid I \mid J \mid K \mid L \mid M \mid \\ & N \mid O \mid P \mid Q \mid R \mid S \mid T \mid U \mid V \mid W \mid X \mid Y \mid Z \quad . \end{aligned}$$

Opmerking. Hiermee is gedefinieerd dat we ons zullen bedienen van een alfabet van 52 verschillende karakters. Let er op:

dat het cijfer 0 en de letters o en O verschillende karakters zijn;

dat het cijfer 1 en de letters l en I verschillende karakters zijn;

dat het cijfer 9 en de letter g verschillende karakters zijn.

Let er voorts op --de volgorde van de alternatieven in een syntaxregel heeft immers geen betekenis-- dat de 52 letters niet twee aan twee aan elkaar zijn toegevoegd, en dat daarom de letterrij "aap" met de letterrijen "Aap" of "AAP" niets te maken heeft. (Einde van Opmerking.)

In onze voorbeeldjes hebben wij zogenaamde lokale variabelen geïntroduceerd, bijvoorbeeld door de declaratie

```
x, y, z: int      ,
```

en passant hun "namen" (namelijk x, y respectievelijk z) invoerend. Zo goed als we de verschijningsvormen van een natuurlijk getal hebben gedefinieerd, gaan we nu die van een naam definiëren.

Opmerking. In de Engelse literatuur is de standaardterm voor naam: "identifier". Wij zullen voor namen de syntax van ALGOL 60 identifiers aanhouden. (Einde van Opmerking.)

```
< naam > ::= < letter >
          | < naam > < letter >
          | < naam > < cijfer >      .
```

Opgave. Ga na, dat er 199888 verschillende namen van 3 karakters bestaan. (Einde van Opgave.)

Nu zijn we toe aan de definitie van de syntax van expressies die toelaatbaar zijn. We beperken ons in dit stadium tot de syntactische eenheid genaamd "integer expressie"; de syntax beschrijft hoe integer expressies worden opgebouwd uit:

- namen en natuurlijke getallen
- additieve operatoren
- multiplicatieve operatoren
- haakjesparen .

Opmerking. Met opzet beperken we ons in dit stadium tot een bescheiden syntax voor integer expressies. Hij zal later nog wel iets worden uitgebreid. (Einde van Opmerking.)

```
< integer expressie >
    ::= < intterm >
        | < addop > < intterm >
        | < integer expressie > < addop > < intterm > .
```

Bovenstaande syntaxregel vertelt ons dat een integer expressie een rij is waarin exemplaren van de syntactische eenheid `addop` en exemplaren van de syntactische eenheid `intterm` elkaar afwisselen en die eindigt op een (exemplaar van de syntactische eenheid) `intterm`. Nu moeten we alleen nog vertellen hoe een `addop` en een `intterm` er uit kunnen zien. Voor de eerste is dat eenvoudiger dan voor de tweede.

```
< addop > ::= + | -

< intterm >
    ::= < intfactor >
        | < intterm > < multop > < intfactor > .
```

De definitie van `addop` is hiermee voltooid: een plusteken of een minteken. De volgende syntaxregel vertelt ons dat een `intterm` bestaat uit één of meer exemplaren van de syntactische eenheid `intfactor`, onderling gescheiden door een exemplaar van

de syntactische eenheid `multop` . En nu moeten we alleen nog maar vertellen hoe een `multop` en een `intfactor` er uit kunnen zien; weer is dit voor de eerste eenvoudiger dan voor de tweede.

`< multop > ::= * | / | div | mod`

`< intfactor >`

`::= < natuurlijk getal >`

`| < naam >`

`| ( < integer expressie > ) .`

En hiermee is onze syntactische definitie van de integer expressie voltooid. Wij zullen een voorbeeld ontleden en aantonen dat

`- ab - x mod 3 + 8 *(y + 1)`

een integer expressie is.

Dit geldt op grond van

(0) `- ab - x mod 3` is een integer expressie

(1) `+` is een `addop` (evident)

(2) `8 *(y + 1)` is een `intterm`

Uitspraak (0) geldt op grond van

(0.0) `- ab` is een integer expressie

(0.1) `-` is een `addop` (evident)

(0.2) `x mod 3` is een `intterm`

Uitspraak (0.0) geldt op grond van

(0.0.0) `-` is een `addop` (evident)

(0.0.1) `ab` is een `intterm`

Uitspraak (0.0.1) geldt op grond van

(0.0.1.0) `ab` is een `intfactor`

Uitspraak (0.0.1.0) geldt op grond van  
(0.0.1.0.0) : ab is een naam

Uitspraak (0.0.1.0.0) geldt op grond van  
(0.0.1.0.0.0) a is een naam  
(0.0.1.0.0.1) b is een letter (evident)

Uitspraak (0.0.1.0.0.0) geldt op grond van  
(0.0.1.0.0.0.0) a is een letter (evident)

Hiermee is uitspraak (0.0) bewezen; (0.1) is evident en (0.2) geldt op grond van

(0.2.0) x is een intterm  
(0.2.1) mod is een multop (evident)  
(0.2.2) 3 is een intfactor

Uitspraak (0.2.0) geldt op grond van  
(0.2.0.0) x is een intfactor

Uitspraak (0.2.0.0) geldt op grond van  
(0.2.0.0.0) x is een naam

Uitspraak (0.2.0.0.0) geldt op grond van  
(0.2.0.0.0.0) x is een letter (evident)

Uitspraak (0.2.0) is hiermee bewezen; (0.2.1) is evident en  
(0.2.2) geldt op grond van  
(0.2.2.0) 3 is een natuurlijk getal

Uitspraak (0.2.2.0) geldt op grond van  
(0.2.2.0.0) 3 is een cijfer (evident)

Hiermee is (0.2.2), en daarmee (0.2) en daarmee (0) bewezen;

(1) is evident en het bewijs van (2) wordt aan de lezer overgelaten.

We suggereren niet dat hij een net zo breedsprakig bewijs levert als wij voor (0) geleverd hebben, maar wel dat hij zich elk appel op de formele syntax realiseert. Wij hebben de ontleding in zulke kleine stapjes laten zien opdat, ten eerste, de lezer zich kan voorstellen, dat dit door een automaat kan geschieden, en, ten tweede, de lezer apprecieert dat een formele definitie van de syntax hiervoor onontbeerlijk is. (Ontwikkeling en eerste implementatie van FORTRAN in het midden van de 50-er jaren hebben meer dan 100 keer zoveel manjaar geleverd als de eerste implementatie van ALGOL 60 in 1960. De twee projecten verschilden genoeg om met de interpretatie van die factor 100 wat voorzichtig te zijn; anderzijds waren ze soortgelijk genoeg om deze factor onder andere te willen verklaren door de omstandigheid dat van FORTRAN elke formele definitie ontbrak.)

Wij vestigen er voorts de aandacht op dat onze syntax van integer expressies expliciet tot uitdrukking brengt dat additieve operatoren zogenaamd "links-associatief" zijn, dat wil zeggen dat bijvoorbeeld een integer expressie als  $-a - b + c$  slechts als integer expressie geldt in de ontleding

$$((-a) - b) + c$$

en dat interpretaties zoals

$$(-a) - (b + c) \quad \text{en} \quad -(a - (b + c))$$

derhalve niet zijn toegestaan. De syntax voor integer expressies doet dus meer dan alleen maar definiëren welke symboolrijen als integer expressies gelden: hij geeft ook aan hoe de expressie moet worden geïnterpreteerd, dat wil zeggen welke constanten en welke tussenresultaten de operanden van welke operatoren dienen te zijn wanneer de expressie wordt berekend.

Opgave. Stel vast waarom  $2k + 1$  niet een syntactisch correcte integer expressie is. (Einde van Opgave.)

Opmerking. Ook de multiplicatieve operatoren zijn links-associatief gedefinieerd, en de vermenigvuldiging is geen prioriteit boven de deling gegeven:  $m / 2 * h$  is dus een afkorting van  $(m / 2) * h$  en *niet* van  $m / (2 * h)$ . Als dat laatste bedoeld is, mogen de haakjes dus niet weggelaten worden. Het advies met haakjesparen niet te karig te zijn geldt echter veel algemener: over welke zonder betekenisverandering kunnen worden weggelaten ontbreekt vaak de consensus. (Einde van Opmerking.)

Rest ons slechts de operatoren te definiëren. Van de additieve operatoren vermelden we slechts dat, indien ze als binaire operatoren voorkomen, met  $+$  de optelling en met  $-$  de aftrekking wordt aangegeven, en dat, indien ze als unaire operator voorkomen --dat wil zeggen als eerste symbool van een integer expressie-- de  $+$  geen effect heeft en de  $-$  tekenwisseling aangeeft. We vertrouwen er hier op dat de lezer weet, wat met optelling, aftrekking en tekenwisseling bedoeld is. We spreiden eenzelfde vertrouwen ten aanzien van de vermenigvuldiging ten toon, wanneer we verklaren dat met  $*$  de vermenigvuldiging van twee gehele getallen wordt aangegeven.

De resterende multiplicatieve operatoren,  $/$ , div en mod zijn zogenaamde "partiële operatoren", dat wil zeggen  $x / y$ ,  $x \text{ div } y$  en  $x \text{ mod } y$  zijn niet voor elk paar gehele waarden  $(x, y)$  gedefinieerd.

Met  $x / y$  wordt het quotiënt van  $x$  en  $y$  aangegeven, dat uit de aard der zaak slechts gedefinieerd is mits  $y \neq 0$ ; voorts spreken wij af ons bij het gebruik van  $x / y$  te zullen beperken tot die situaties, waarin  $x$  een geheel aantal malen  $y$  is.

Voor  $x \text{ div } y$  en  $x \text{ mod } y$  geldt alleen de beperking  $y \neq 0$ :  
 $x \text{ div } y = q$  en  $x \text{ mod } y = r$ , waar de gehele  $q$  en  $r$  voldoen aan

$$x = q \cdot y + r \wedge 0 \leq r < \text{abs}(y) \quad .$$

Merk op:

$$\begin{aligned} x \text{ mod } y &= x \text{ mod } (-y) \\ (-x) \text{ div } y &\neq -x \text{ div } y, \text{ tenzij } x \text{ mod } y = 0 \\ (x + y) \text{ mod } y &= x \text{ mod } y \\ (x + y) \text{ div } y &= 1 + x \text{ div } y \end{aligned} \quad .$$

En hiermee is voorlopig de beschrijving voltooid van wat wij als "toelaatbare expressies" hadden aangeduid.

## Concatenatie van statements

Tot nog toe zijn we alleen de assignment statement van de vorm  $x := E$  tegengekomen, en de programmaatjes die we tot dusver kunnen opschrijven zijn in tweeërlei opzicht erg beperkt. Ten eerste blijft de toestandswijziging die zij kunnen bewerkstelligen beperkt tot wijziging van de waarde van één variabele van de toestandsruimte, ten tweede is zijn nieuwe waarde beperkt tot wat wij met een toelaatbare expressie kunnen uitdrukken. Wij zullen nu laten zien, hoe de eerste beperking ondervangen wordt.

Overal, waar wij een statement mogen schrijven, mogen wij ook een zogenaamde "statement list" schrijven

```
< statement list >
 ::= < statement >
    | < statement list >; < statement >
```

dat wil zeggen een lijst van 1 of meer statements, in het laatste geval onderling verbonden door de puntkomma. De puntkomma die tussen twee opeenvolgende statements wordt geschreven verbindt die twee statements in die zin dat de uitvoering van de linker statement door die van de rechter statement gevolgd dient te worden, en dat de eindconditie van de linker statement geïdentificeerd wordt met de beginconditie van de rechter statement.

Aan het einde van ons hoofdstuk over de assignment statement hebben we de algemene structuur van onze uitspraken over een enkele assignment statement gegeven. Wij herhalen:

$$I[ x, y, z: \text{int } \{Q\}; x := E0 \{R\} ]I$$

betekent dat als voor de uitvoering van  $x := E0$  voldaan is aan  $Q$ , na afloop van de uitvoering van  $x := E0$  de toestand aan  $R$  voldoet; deze uitspraak is juist als  $Q$  gelijk is aan  $R_{E0}^x$ , dat wil zeggen de conditie  $R$  waarin voor  $x$  de expressie  $E0$  --zonodig tussen haakjes, maar dat zullen we nu niet meer steeds herhalen-- is gesubstitueerd.

Met  $P = Q_{E1}^y$  is

$$I[ x, y, z: \text{int } \{P\}; y := E1 \{Q\} ]I$$

evenzo een juiste uitspraak, dit maal over de assignment statement  $y := E1$ . Het postulaat over de concatenatie behelst dat deze twee uitspraken gecombineerd mogen worden tot

$$I[ x, y, z: \text{int } \{P\}; y := E1 \{Q\}; x := E0 \{R\} ]I$$

of, als we  $Q$  elimineren door hem te verzwijgen, tot de uitspraak over " $y := E1; x := E0$ "

$$I[ x, y, z: \text{int } \{P\}; y := E1; x := E0 \{R\} ]I \quad .$$

We kunnen nu ook nagaan, welke uitspraken we kunnen doen als we de twee statements verwisselen, dat wil zeggen uitspraken van de vorm

$$| [ x, y, z: \text{int } \{P'\}; x := E0; y := E1 \{R\} ] | \quad .$$

Werkend van rechts naar links vormen we eerst  $Q' = R_{E1}^y$ , en vervolgens  $P' = Q'_{E0}^x$ . Over het algemeen is  $P \neq P'$  --namelijk als  $x$  in  $E1$  voorkomt, of  $y$  in  $E0$  --. Met andere woorden concatenatie van statements is in het algemeen *niet* commutatief.

Als voorbeeld zullen we  $P$  afleiden, opdat de uitspraak

$$| [ x, y, z: \text{int } \{P\} \\ ; x := x + y; y := x - y; x := x - y \\ \{x = X \wedge y = Y \wedge z = Z\} \\ ] |$$

geldt.

Om te beginnen voeren we de --nu twee-- verzwegen tussencondities in:

$$| [ x, y, z: \text{int } \{P\} \\ ; x := x + y \{Q1\} \\ ; y := x - y \{Q0\} \\ ; x := x - y \\ \{x = X \wedge y = Y \wedge z = Z\} \\ ] | \quad .$$

Werkend van achteren naar voren vinden we door in de eindconditie  $x - y$  voor  $x$  te substitueren

$$Q0: \quad x - y = X \wedge y = Y \wedge z = Z \quad .$$

Daarin  $y$  door  $x - y$  vervangend vinden we

$$Q1: \quad x - (x - y) = X \wedge x - y = Y \wedge z = Z \quad ,$$

en na vereenvoudiging

$$Q1: \quad y = X \wedge x - y = Y \wedge z = Z \quad .$$

Daarin  $x$  vervangend door  $x + y$  vinden we

$$P: \quad y = X \wedge (x + y) - y = Y \wedge z = Z \quad ,$$

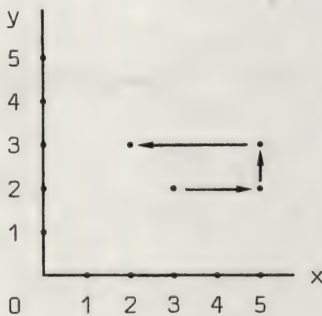
en na vereenvoudiging

$$P: \quad y = X \wedge x = Y \wedge z = Z \quad .$$

Vergelijken we  $P$  met onze eindconditie, dan zien we dat onze "gedurige concatenatie"

$$"x := x + y; y := x - y; x := x - y"$$

de waarden van  $x$  en  $y$  verwisselt en alle andere variabelen ongemoeid laat.



In het bovenstaande plaatje --een projectie van de toestandsruimte op het  $x, y$ -vlak-- hebben we het pad aangegeven voor het speciale geval  $X = 2 \wedge Y = 3$ . Het plaatje is alleen gegeven ter illustratie van het feit dat de statement concatenatie ons in staat stelt programma's te construeren die de gewenste toestandsverandering niet in één klap, maar in een aantal opeenvolgende stappen bewerkstelligen. In deze beeldspraak wordt de berekening een pad door de toestandsruimte dat leidt van beginpunt tot eindpunt. We zullen later zien dat in de praktijk deze paden in zeer veel stappen plegen te worden afgelegd.

Wij vestigen er de aandacht op dat het bovenstaande plaatje

*alleen* maar gegeven is ter illustratie van de beeldspraak. In de praktijk van het programmeren worden dergelijke plaatjes *nooit* getekend. Ze zouden het praktisch bezwaar hebben verschrikkelijk ingewikkeld te worden; ze hebben het principiële bezwaar slechts op een heel specifiek geval betrekking te hebben (zoals hier op het geval  $X = 2 \wedge Y = 3$  ).

Opmerking. Ook bovenstaande gedurige concatenatie is louter om illustratieve redenen ten tonele gevoerd. Wij zullen later realistischere oplossingen tegenkomen om twee variabelen van waarde te laten verwisselen. (Einde van Opmerking.)

## De alternatieve statement

In het vorige hebben we gezien dat voor een gegeven programma het pad door de toestandsruimte afhangt van de begintoestand, maar dat, zolang we alleen assignment statements en hun concatenatie tot onze beschikking hebben, onafhankelijk van de begintoestand altijd dezelfde serie statements uitgevoerd zal worden. We zullen nu laten zien hoe bij een gegeven programma de begintoestand ook kan bepalen welke statements uitgevoerd zullen worden. Dat we deze grotere flexibiliteit nodig hebben, ontdekken we bij de bestudering van de functionele specificatie van het programma "largest"

```
[[ x, y, z: int {x = X  $\wedge$  y = Y}
  ; largest {x = X  $\wedge$  y = Y  $\wedge$  z = max(x, y)}
]] .
```

Het probleem zou triviaal zijn als "max(x, y)" voorkwam onder de toelaatbare expressies, want dan zou "z := max(x, y)" de functionele specificatie van largest bevredigen. Maar aangezien

$\max(x, y)$  niet onder de toelaatbare expressies voorkomt, moeten we iets anders doen.

Opmerking. Er bestaan programmeertalen die de programmeur toestaan zelf " $\max(x, y)$ " aan de collectie toelaatbare expressies toe te voegen. De toevoeging vergt dan evenwel een oplossing van het probleem dat hier gesteld wordt: in termen van de reeds toelaatbare expressies een programma schrijven dat aan de functionele specificatie van `largest` voldoet. (Einde van Opmerking.)

Wij merken om te beginnen op dat we met de eindconditie voor `largest` met behulp van het postulaat van de assignment statement voor `z := x` afleiden

```

| [ x, y, z: int {x = X ∧ y = Y ∧ x = max(x, y)}
  ; z := x {x = X ∧ y = Y ∧ z = max(x, y)}
| ] .

```

Voorts merken we op dat voor alle  $x$  en  $y$

$$(x = \max(x, y)) = (x \geq y)$$

--dat wil zeggen  $x = \max(x, y)$  en  $x \geq y$  zijn òf beide `true` òf beide `false` -- ; deze gelijkheid stelt ons in staat uit de beginconditie de "ontoelaatbare" expressie  $\max(x, y)$  te elimineren:

```

| [ x, y, z: int {x = X ∧ y = Y ∧ x ≥ y}
  ; z := x {x = X ∧ y = Y ∧ z = max(x, y)}
| ] .

```

Op dezelfde manier leiden we af

```

| [ x, y, z: int {x = X ∧ y = Y ∧ y ≥ x}
  ; z := y {x = X ∧ y = Y ∧ z = max(x, y)}
| ] .

```

Aan de laatste term van de begintoestand zien we dat in sommige begintoestanden --namelijk als  $x \geq y$  -- de assignment statement  $z := x$  het gewenste effect bewerkstelligt en dat in (grotendeels) andere begintoestanden --namelijk als  $y \geq x$  -- de assignment statement  $z := y$  dat doet. Omdat  $x \geq y \vee y \geq x$  altijd geldt, voorzien beide assignments samen in alle gevallen, en we zouden ze graag combineren, de keuze bij uitvoering van de begintoestand af latend hangen. Dit kan met behulp van de zogenaamde alternatieve statement

```

[[ x, y, z: int {x = X  $\wedge$  y = Y}
  ; if  $x \geq y \rightarrow z := x$  []  $y \geq x \rightarrow z := y$  fi
    {x = X  $\wedge$  y = Y  $\wedge$  z = max(x, y)}
]] .

```

Deze vorm van combinatie is niet tot 2 programma's beperkt: het mag voor elk eindig aantal. We zullen het algemene schema aan het geval van 3 stuks illustreren --wederom met een toestandsruimte opgespannen door  $x$ ,  $y$  en  $z$  -- . Het is het postulaat van de alternatieve statement.

De drie uitspraken

```

[[ x, y, z: int {P  $\wedge$  B0}; S0 {R} ]] ,
[[ x, y, z: int {P  $\wedge$  B1}; S1 {R} ]] en
[[ x, y, z: int {P  $\wedge$  B2}; S2 {R} ]]

```

wettigen te zamen de uitspraak

```

[[ x, y, z: int {P  $\wedge$  (B0  $\vee$  B1  $\vee$  B2)}
  ; if B0  $\rightarrow$  S0
    [] B1  $\rightarrow$  S1
    [] B2  $\rightarrow$  S2
    fi {R}
]] .

```

Opmerking. Rechtstreekse toepassing van bovenstaande regel had in ons voorbeeld de beginconditie

$$\{x = X \wedge y = Y \wedge (x \geq y \vee y \geq x)\}$$

opgeleverd, maar omdat de laatste term, zoals opgemerkt, altijd geldt kan hij zonder betekenisverandering worden weggelaten.

(Einde van Opmerking.)

Wij zullen eerst onze syntax passend uitbreiden.

< statement >

::= skip

| < assignment statement >

| < alternatieve statement >

< alternatieve statement >

::= if < guarded command set > fi

< guarded command set >

::= < guarded command >

| < guarded command set > [] < guarded command >

< guarded command >

::= < guard > → < statement list >

Een guard is een zogenaamde Boolse expressie; de preciese definitie van welke Boolse expressies toelaatbaar zijn wordt voorshands nog even uitgesteld.

Wij vestigen er de aandacht op dat uit het postulaat van de alternatieve statement volgt dat de volgorde waarin de guarded commands binnen het hakenpaar if ... fi worden opgesomd, niet ter zake doet. (Om die reden is de syntactische eenheid dan ook "guarded command set" en niet "guarded command list" genoemd.)

Opgave. Leidt uit het postulaat van de alternatieve statement af dat het, hoewel toelaatbaar, geen zin heeft in een guarded command

set een bepaalde guarded command meer dan eens te laten voorkomen.  
(Einde van Opgave.)

Operationeel behelst een guarded command van de vorm  $B \rightarrow S$  dat de statement list  $S$  slechts voor uitvoering in aanmerking komt in die begintoestanden waarin de guard  $B$  geldt.

Operationeel behelst de alternatieve statement dat precies 1 van de guarded commands voor uitvoering geselecteerd wordt, en wel eentje waarvan de guard aanvankelijk true is. Zijn in de begintoestand van een alternatieve statement alle guards false, dan geldt dit als een programmafout (die in elke redelijke implementatie gehonoreerd wordt met een onmiddellijke onderbreking van de uitvoering van het programma). De programmeur heeft derhalve de plicht aan te tonen dat vóór elke alternatieve statement ten minste 1 van zijn guards geldt. (In ons voorbeeld voor largest was deze plicht triviaal.) Een en ander heeft tot consequentie dat er voor alternatieve statements met maar 1 guarded command vrij weinig emplooi is.

Wij vestigen er de aandacht op dat in die gevallen waarin méér dan 1 guard geldt, volledig in het midden wordt gelaten welke statement list met geldende guard voor uitvoering wordt gekozen. Bij onze tekst voor largest

$$\text{if } x \geq y \rightarrow z := x \quad \square \quad y \geq x \rightarrow z := y \quad \text{fi}$$

en begintoestand  $x = 7 \wedge y = 7$  gelden beide guards. Het zal ons inderdaad een zorg zijn of  $z = 7$  tot stand gebracht wordt door  $z := x$  of door  $z := y$ .

Het feit dat, indien er keuze mogelijk is doordat de guards elkaar niet uitsluiten, deze keuze volledig vrijgelaten wordt introduceert zogenaamd nondeterminisme. In het geval van nondeter-

minisme heeft het pad door de toestandsruimte --en daarmee de eindtoestand-- niet meer uniek door de begintoestand bepaald te zijn, zoals geïllustreerd wordt door de statement waarvan de specificatie uitdrukt dat zijn uitvoering  $x$  al dan niet van teken wisselt:

```

| [ x: int
    {x = X}
  ; if true  $\rightarrow$  x := - x  [] true  $\rightarrow$  skip fi
    {abs(x) = abs(X)}
| | .

```

Noodzaak voor het schrijven van een nondeterministisch programma is er nooit. Uit een nondeterministisch programma dat aan een bepaalde specificatie voldoet laat zich namelijk altijd een deterministisch programma afleiden dat dezelfde specificatie bevredigt.

De functionele specificatie van statement  $S$  zij gegeven door beginconditie  $P$  en eindconditie  $Q$ , precieser

```

| [ x, y, z: int {P}; S {Q} ] | .

```

Aan deze specificatie zij aantoonbaar voldaan door een  $S$  van de vorm

```

if  $B_0 \rightarrow S_0$  []  $B_1 \rightarrow S_1$  fi .

```

Vanwege het postulaat van de alternatieve statement betekent dit de juistheid van de volgende uitspraken:

```

 $P \Rightarrow (B_0 \vee B_1)$ 
| [ x, y, z: int {P  $\wedge$   $B_0$ };  $S_0$  {Q} ] |
| [ x, y, z: int {P  $\wedge$   $B_1$ };  $S_1$  {Q} ] | .

```

Hieruit volgt echter (in volgorde)

```

 $P \Rightarrow (B_0 \vee (B_1 \wedge \neg B_0))$ 
| [ x, y, z: int {P  $\wedge$   $B_0$ };  $S_0$  {Q} ] |
| [ x, y, z: int {P  $\wedge$   $B_1 \wedge \neg B_0$ };  $S_1$  {Q} ] | ,

```

waaruit wij, op grond van het postulaat van de alternatieve statement, concluderen dat wij voor  $S$  ook hadden mogen kiezen

$$\underline{\text{if}} \ B0 \rightarrow S0 \ \square \ B1 \wedge \neg B0 \rightarrow S1 \ \underline{\text{fi}} \ ,$$

dat wil zeggen de guard van de ene guarded command mag straffeloos worden versterkt met de neveneis dat de andere guarded command niet voor uitvoering in aanmerking komt. Maar nu hebben we een alternatieve statement waarin de guards elkaar uitsluiten en die derhalve deterministisch is. Uit onze gevonden oplossing voor `largest` op deze wijze het nondeterminisme uitbannen leidt tot de deterministische oplossingen

$$\underline{\text{if}} \ x \geq y \rightarrow z := x \ \square \ y > x \rightarrow z := y \ \underline{\text{fi}}$$

en

$$\underline{\text{if}} \ x > y \rightarrow z := x \ \square \ y \geq x \rightarrow z := y \ \underline{\text{fi}} \ .$$

Opgave. Toon aan dat de volgende alternatieve statement voldoet aan de specificatie van `largest` :

$$\begin{aligned} &\underline{\text{if}} \ x > y \rightarrow z := x \\ &\quad \square \ x = y \rightarrow z := (x + y) / 2 \\ &\quad \square \ x < y \rightarrow z := y \\ &\underline{\text{fi}} \qquad \qquad \qquad (\text{Einde van Opgave.}) \end{aligned}$$

Zal in dit college de noodzaak nondeterministische programma's te ontwerpen dus niet optreden, wel is het uiterst gewenst dat wij leren hoe wij over nondeterministische programma's moeten redeneren. Tijdens het ontwerpproces moeten wij kunnen redeneren over een half-af programma, waarin allerlei detailbeslissingen nog niet zijn genomen: zo'n half-af programma heeft vaak de vorm van een non-deterministisch programma.

## Toelaatbare Boolse expressies

Wij zullen nu de syntax geven voor de Boolse expressies die als guard mogen optreden. De syntax zal weer in BNF gegeven worden. In opbouw lijkt hij op die van de integer expressie --de manier waarop haakjes worden geïntroduceerd is bijvoorbeeld strikt analoog-- ; de veelheid van operatoren maakt hem evenwel iets ingewikkelder.

< Boolse expressie >

```
 ::= < disjunctie >
    | < disjunctie >  $\equiv$  < disjunctie >
    | < disjunctie >  $\neq$  < disjunctie >
```

< disjunctie >

```
 ::= < conjunctie >
    | < disjunctie >  $\vee$  < conjunctie >
    | < disjunctie > cor < conjunctie >
```

< conjunctie >

```
 ::= < Boolse term >
    | < conjunctie >  $\wedge$  < Boolse term >
    | < conjunctie > cand < Boolse term >
```

< Boolse term >

```
 ::= < Boolse primary >
    |  $\neg$  < Boolse primary >
```

< Boolse primary >

```
 ::= true
    | false
    | < naam >
    | < integer expressie > < relop > < integer expressie >
    | (< Boolse expressie >)
```

< relop > ::= < |  $\leq$  |  $>$  |  $\geq$  | = |  $\neq$

Wij mogen desgewenst schrijven: and voor  $\wedge$ , or voor  $\vee$  en non voor  $\neg$ . (Vooral bij het gebruik van een schrijfmachine kan dit voordelen hebben.)

De operatoren cand en cor staan voor de zogenaamde "conditionele and" en "conditionele or". Als beide operanden  $a$  en  $b$  gedefinieerd zijn, geldt

$$a \text{ cand } b \equiv a \wedge b, \text{ en}$$

$$a \text{ cor } b \equiv a \vee b.$$

In tegenstelling tot de operatoren  $\wedge$  en  $\vee$ , die slechts gedefinieerd zijn mits beide operanden gedefinieerd zijn, is

$$\text{false cand } b \equiv \text{false}, \text{ en}$$

$$\text{true cor } b \equiv \text{true},$$

ook in die gevallen, waarin  $b$  ongedefinieerd is. In tegenstelling tot  $\wedge$  en  $\vee$  zijn cand en cor derhalve niet commutatief. Mits we de volgorde der operanden niet verwisselen, geldt voor hen de Wet van de Morgan wel, dat wil zeggen

$$\neg(a \text{ cand } b) \equiv (\neg a) \text{ cor } (\neg b).$$

Wij merken op dat  $a \wedge b \vee c$  slechts op één manier ontleed kan worden, namelijk als de disjunctie van  $a \wedge b$  en  $c$ . (De ontleedpoging tot de conjunctie van  $a$  en  $b \vee c$  faalt, omdat blijkens de syntax de rechteroperand van  $\wedge$  een Boolse term moet zijn en  $b \vee c$  niet de gedaante heeft van een Boolse term.) Onze syntax brengt dus de gebruikelijke conventie tot uitdrukking dat  $\wedge$  hogere prioriteit (= "greater binding power") heeft dan  $\vee$ . Het verdient in het algemeen de voorkeur niet al te krenterig met hakenparen te zijn. (Merk op, dat we in plaats van bovenstaande weergave van de Wet van de Morgan ook

$$\neg(a \text{ cand } b) \equiv \neg a \text{ cor } \neg b$$

hadden mogen schrijven.) Het is onverstandig, zich op de hogere

prioriteit van  $\wedge$  ten opzichte van  $\vee$  beroepend, haakjes te besparen waar dit de symmetrie verstoort als in de formulering van de stelling dat voor alle Boolse  $p$ ,  $q$  en  $r$  geldt

$$\begin{aligned} (p \vee q) \wedge (q \vee r) \wedge (r \vee p) &\equiv \\ (p \wedge q) \vee (q \wedge r) \vee (r \wedge p) &\quad . \end{aligned}$$

Opgave. Bewijs de zojuist gegeven stelling. (Einde van Opgave.)

Opmerking. Mits van alle variabelen en constanten --en dat zulks bij ons het geval zal zijn, zal later blijken-- het type ondubbelzinnig vastligt, wordt geen dubbelzinnigheid geïntroduceerd wanneer de Boolse gelijkheid " $\equiv$ " wordt aangegeven met hetzelfde symbool " $=$ " dat ook voor de gelijkheid van integers gebruikt wordt. Als bijvoorbeeld ondubbelzinnig vastligt, dat waar

$$x = y = b$$

staat,  $x$  en  $y$  voor integer waarden staan en  $b$  voor een Boolse waarde, kan dit slechts als

$$(x = y) \equiv b$$

geïnterpreteerd worden. Er zijn op dit gebied geen alom aanvaarde conventies, en dit gebrek aan consensus zullen wij in dit college opvangen door, zoals gezegd, met haakjes niet te krenterig te zijn en door voor de Boolse gelijkheid ten overvloede het speciale symbool " $\equiv$ " te gebruiken. De prijs is gering daar we erg ingewikkelde Boolse expressies toch maar zelden zullen tegenkomen. (Einde van Opmerking.)

Analoog aan variabelen "van het type integer" kunnen we ook variabelen "van het type boolean" declareren: zij hebben slechts twee mogelijke waarden, "true" respectievelijk "false" genaamd. Voor declaraties geldt de syntax

```
< declaratie > ::= < naamlijst > : < type >

< naamlijst > ::= < naam >
                | < naamlijst > , < naam >

< type > ::= int | bool

< declaratie list >
    ::= < declaratie >
       | < declaratie list > ; < declaratie > .
```

De namen in de naamlijst worden onderling door komma's gescheiden; opeenvolgende declaraties worden gescheiden --dan wel verbonden!-- door puntkomma's. De beschrijving van een toestandsruimte kan dus bijvoorbeeld beginnen met

```
[[ x, y: int; b: bool .
```

Analoog aan de assignment van het type integer hebben we ook de assignment van het type boolean, bijvoorbeeld

```
b := x ≥ y
```

welks uitvoering in de nieuwe waarde van  $b$  vastlegt of  $x \geq y$  geldt.

Historische opmerking. Gedurende de eerste 15 jaar is programma-uitvoering begrepen als een combinatie van "het berekenen van getallen" en "het testen van condities", en terwijl het resultaat van zo'n (numerieke) berekening in register of geheugen werd gevormd en achtergelaten voor later gebruik, werd de uitslag van de test van een conditie onmiddellijk gebruikt om (als in een alternatieve statement) de verdere afloop van de berekening te beïnvloeden. Het is de verdienste van ALGOL 60 geweest door de introductie van variabelen van het type boolean duidelijk te maken dat het testen van een conditie beter begrepen kan worden als een berekening, maar nu niet de berekening van een getal, maar de berekening van een zogenaamde "truth value". Deze generalisatie van

het begrip berekening is een heel belangrijke bijdrage: het bewijzen van een stelling kan nu worden gezien als het aantonen dat de berekening van een propositie de waarde `true` oplevert. Hoewel wij in onze programma's slechts een bescheiden aantal variabelen van het type `boolean` zullen tegenkomen, mag het type `boolean` daarom in geen enkele inleiding tot het programmeren ontbreken. De ouderwetse gewoonte, die wij in de electrotechniek nog wel tegenkomen, om de waarden `true` en `false` te identificeren met de integers `1` respectievelijk `0` verdient dan ook geen navolging: hij sticht alleen maar verwarring. (Einde van Historische opmerking.)

## De repetitieve statement

Aan ons arsenaal mogelijkheden om programma's op te bouwen ontbreekt nog iets heel essentieels. Zolang we alleen concatenatie hadden bestond de uitvoering van een programma uit de uitvoering van evenveel statements als wij achter elkaar hadden neergeschreven. Het effect van de alternatieve statement is dat statements, hoewel neergeschreven, niet voor uitvoering in aanmerking komen doordat een ander alternatief gekozen wordt.

Er zijn echter rekenopgaven die intrinsiek heel bewerkelijk zijn. (Het zijn juist die opgaven, waaraan de computer zijn bestaansrecht ontleent! Het zijn precies die opgaven, die de grote snelheid van rekenorganen en geheugens rechtvaardigen.) Die rekenopgaven zijn zo bewerkelijk doordat de gewenste toestandsverandering alleen bewerkstelligd kan worden door een heel lang pad, dat uit heel veel stapjes bestaat.

Men moet zich realiseren dat machines die in luttele seconden miljoen assignment statements hebben uitgevoerd helemaal geen uitzondering meer zijn. Het is evident dat het programmeren voor zo'n machine onbegonnen werk zou zijn, als we, om hem luttele seconden aan het werk te houden, eerst een miljoen statements zouden moeten schrijven. Het moet mogelijk zijn om korte programma's te schrijven waarmee lange rekenprocessen corresponderen; zonder die mogelijkheid zouden die lange rekenprocessen niet te realiseren zijn.

Laat ons teruggrijpen op de formele specificatie van euclid :

```

[[ x, y: int
   {X = ggd(x, y)  ∧  x > 0  ∧  y > 0}
; euclid
   {x = X  ∧  y = X}
]] .

```

Omdat  $\text{ggd}(X, X) = X$  en  $X > 0$ , geldt in de eindtoestand de beginconditie nog steeds; in de eindtoestand geldt bovendien  $x = y$ . Met  $P$  gedefinieerd als

$$P: \quad X = \text{ggd}(x, y) \wedge x > 0 \wedge y > 0$$

mogen we euclid ook specificeren door

```

[[ x, y: int {P}; euclid {P ∧ x = y} ]]

```

omdat, omgekeerd, uit  $P \wedge x = y$  de oorspronkelijke eindconditie volgt. (Ga dit na.) Deze laatste functionele specificatie werpt een nieuw licht op de taak van euclid: zonder de geldigheid van  $P$  te verstoren, dient euclid  $x = y$  te bewerkstelligen.

En als we dat niet in één klap kunnen, wel, dan zijn we tevreden als we zonder verstoring van  $P$  een stap in de goede richting kunnen doen. Die stap kan dan herhaald worden totdat het einddoel  $x = y$  is bereikt. Als het einddoel  $x = y$  nog niet bereikt is,

geldt  $x > y$  of  $y > x$ . Omdat de ggd van twee getallen gelijk is aan die van één van beide en hun verschil, gelden de volgende twee uitspraken

```

| [ x, y: int {P ∧ x > y}; x := x - y {P} ] |
| [ x, y: int {P ∧ y > x}; y := y - x {P} ] | .

```

Een oplossing voor euclid is nu

```

do x > y → x := x - y
  [] y > x → y := y - x
od .

```

De volgorde waarin de guarded commands binnen het hakenpaar do...od worden opgesomd doet wederom niet ter zake. Operationeel geldt weer dat een guarded command van de vorm  $B \rightarrow S$  behelst dat de statement list  $S$  slechts voor uitvoering in aanmerking komt in die begintoestanden waarvoor de guard  $B$  geldt.

Het hakenpaar do...od vormt een zogenaamde repetitieve statement. Een repetitieve statement eindigt slechts in toestanden waarin geen van zijn guards geldt. Operationeel behelst de uitvoering van een repetitieve statement het volgende.

Als alle guards false bevonden worden, reduceert de verdere uitvoering van de repetitieve statement zich tot een skip, anders wordt een guarded command, waarvan de guard true is, uitgevoerd, waarna dit proces herhaald wordt. (Vandaar de naam "repetitieve statement".)

Zo is bijvoorbeeld de repetitieve statement do  $B \rightarrow S$  od equivalent met het langere programma

```

if ¬B → skip
  [] B → S; do B → S od
fi

```

Opmerking. Bovenstaande equivalentie kan gebruikt worden voor de definitie van do  $B \rightarrow S$  od . (Einde van Opmerking.)

Stel, dat euclid uitgevoerd wordt in de begintoestand  $(x, y) = (8, 6)$  . De eerste guard geldt en uitvoering van  $x := x - y$  leidt tot de toestand  $(2, 6)$  ; dan geldt de tweede guard en uitvoering van  $y := y - x$  leidt tot  $(2, 4)$  ; weer geldt de tweede guard en uitvoering van  $y := y - x$  leidt tot  $(2, 2)$  ; omdat nu beide guards false zijn is hiermee de uitvoering van de repetitieve statement voltooid.

De passende uitbreiding van onze syntax is

```
< statement > ::= skip
                | < assignment statement >
                | < alternatieve statement >
                | < repetitieve statement >

< repetitieve statement >
    ::= do < guarded command set > od .
```

Het postulaat van de repetitieve statement zullen wij formuleren voor het geval van 2 guarded commands (de generalisatie tot meer of minder aan de lezer overlatend).

De uitspraken

```
[[ x, y, z: int {P ∧ B0}; S0 {P} ]]
[[ x, y, z: int {P ∧ B1}; S1 {P} ]]
```

wettigen te zamen de uitspraak

```

[[ x, y, z: int
   {P}
   ; do B0  $\rightarrow$  S0
     [] B1  $\rightarrow$  S1
   od
   {P  $\wedge$   $\neg$ B0  $\wedge$   $\neg$ B1}
]]

```

mits de repetitieve statement eindigt. (Op deze extra voorwaarde zullen wij later terugkomen.) In een beroep op het postulaat van de repetitieve statement wordt het hier met P aangeduide predicaat "de invariant" genoemd.

Wij gaan nu het programma euclid, dat de ggd van twee positieve getallen uitrekent, uitbreiden tot het programma euclid-plus, dat tevens hun kgv (= kleinste gemene veelvoud) berekent. Met dezelfde P als weleer:

P:  $X = \text{ggd}(x, y) \wedge x > 0 \wedge y > 0$

luidt de voorlopige --waarom voorlopig zal straks blijken-- functionele specificatie van euclidplus

```

[[ x, y, u, v: int
   {P  $\wedge$  Y = kgv(x, y)}
   ; euclidplus
   {x = X  $\wedge$  y = Y}
]] .

```

Wij definiëren Q door

Q:  $P \wedge x \cdot u + y \cdot v = 2 \cdot X \cdot Y$  ,

en stellen om te beginnen vast

```

0)  |[ x, y, u, v: int
      {P  $\wedge$  Y = kgv(x, y)}
      ; u:= y; v:= x
      {Q}
    ]| ,

```

hetwelk berust op de stelling dat voor positieve x en y

$$\text{ggd}(x, y) \cdot \text{kgv}(x, y) = x \cdot y .$$

Voorts stellen wij vast

```

1)  |[ x, y, u, v: int
      {Q  $\wedge$  x > y}
      ; x:= x - y; v:= v + u
      {Q}
    ]|

```

omdat  $x := x - y; v := v + u$  --ga dit na!-- de waarde van de som  $x \cdot u + y \cdot v$  niet wijzigt. Om redenen van symmetrie geldt dus tevens

```

2)  |[ x, y, u, v: int
      {Q  $\wedge$  y > x}
      ; y:= y - x; u:= u + v
      {Q}
    ]|

```

en volgens het postulaat van de repetitieve statement mogen we 1) en 2) onder voorbehoud van beëindiging combineren tot

```

3)  |[ x, y, u, v: int {Q}
      ; do x > y  $\rightarrow$  x:= x - y; v:= v + u
        || y > x  $\rightarrow$  y:= y - x; u:= u + v
      od {Q  $\wedge$  x = y}
    ]|

```

--dit omdat  $(\neg x > y \wedge \neg y > x) \equiv (x = y)$  -- .

Omdat  $P \wedge x = y$  de conclusie  $x = X \wedge y = X$  wettigt,

wettigt  $Q \wedge x = y$  de conclusie  $X \cdot (u + v) = 2 \cdot X \cdot Y$  oftewel  $Y = (u + v) / 2$ , waaruit wij mogen concluderen

```
4)  |[ x, y, u, v: int
      {Q  $\wedge$  x = y}
      ; y := (u + v) / 2
      {x = X  $\wedge$  y = Y}
    ]| .
```

Combinatie van 0) , 3) en 4) levert dan (vrij volledig geannoteerd)

```
|[ x, y, u, v: int {P  $\wedge$  kgv(x, y) = Y}
  ; u := y; v := x {Q}
  ; do x > y  $\rightarrow$  x := x - y; v := v + u
    || y > x  $\rightarrow$  y := y - x; u := u + v
  od {Q  $\wedge$  x = y}
  ; y := (u + v) / 2 {x = X  $\wedge$  y = Y}
]| .
```

Vergelijking van bovenstaande uitspraak met de voorlopige specificatie van euclidplus leert ons dat

```
u := y; v := x
; do x > y  $\rightarrow$  x := x - y; v := v + u
  || y > x  $\rightarrow$  y := y - x; u := u + v
od; y := (u + v) / 2
```

aan de voorlopige specificatie van euclidplus voldoet.

Wij hebben onze oplossing voor euclidplus om twee heel verschillende redenen ten tonele gevoerd. Ten eerste omdat het de macht van het postulaat van de repetitieve statement aardig illustreert: zonder kennis van de invariant  $Q$  is het niet zo makkelijk in te zien dat in ons laatste programma na afloop van de repetitie het gemiddelde van  $u$  en  $v$  gelijk is aan het kleinste gemene

veelvoud van de aanvangswaarden van  $x$  en  $y$ . Ten tweede omdat de voorlopige functionele specificatie van `euclidplus` een anomalie vertoont: de eerste regel luidt

```
|[ x, y, u, v: int
```

maar --zie  $P$  -- variabelen  $u$  en  $v$  komen noch in de beginconditie, noch in de eindconditie van de functionele specificatie van `euclidplus` voor! Je zou daarom zeggen, dat ze in de functionele specificatie van `euclidplus` helemaal niet zouden moeten voorkomen, dat wil zeggen dat de definitieve functionele specificatie zou moeten luiden:

```
|[ x, y: int
  {P  $\wedge$  Y = kgv(x, y)}
; euclidplus
  {x = X  $\wedge$  y = Y}
]|
```

## Een intermezzo over binnenblokken

Tot nog toe zijn we de haken "`|[`" en "`]|`" slechts tegengekomen als openings- respectievelijk sluitingssymbool van functionele specificaties. Hun functie was daar de tekstuele begrenzing van het geldigheidsbereik van de namen der variabelen die gebruikt werden om de toestandsruimte op te spannen, waarin de functionele specificatie begrepen moest worden.

We introduceren nu hetzelfde hakenpaar en de declaratie van lokale variabelen als onderdeel niet van functionele specificaties, maar van *programmeerteksten*.

Wij breiden daartoe onze syntax uit:

```
< statement > ::= skip
                | < assignment statement >
                | < alternatieve statement >
                | < repetitieve statement >
                | < binnenblok >

< binnenblok >
    ::= [ [ < declaratie list > ; < statement list > ] ] .
```

Het programma dat aan de definitieve functionele specificatie van euclidplus voldoet heeft de vorm van een binnenblok en ziet er als volgt uit:

```
[ [ u, v: int
  ; u:= y; v:= x
  ; do x > y → x:= x - y; v:= v + u
    [ y > x → y:= y - x; u:= u + v
    od
  ; y:= (u + v)/ 2
] ] .
```

Het binnenblok heeft een heel duidelijke operationele betekenis. Als het wordt geactiveerd --men zegt ook wel: "bij binnenkomst"-- wordt de er buiten geldende toestandsruimte uitgebreid met de variabelen die in de aanhef van het binnenblok zijn gedeclareerd; in de aldus uitgebreide toestandsruimte worden de op de declaratie volgende statements uitgevoerd. De voltooiing van de uitvoering van het binnenblok bestaat uit het weer ongedaan maken van de tijdelijke uitbreiding van de toestandsruimte.

In ons voorbeeld wordt de twee-dimensionale toestandsruimte (met coördinaten  $x$  en  $y$ ) bij binnenkomst in het binnenblok tot een vier-dimensionale uitgebreid (namelijk met  $u$  en  $v$ ). Het

inwendige beschrijft een berekening die zich in deze vier-dimensionale toestandsruimte afspeelt. Na de uitvoering van de assignment statement  $y := (u + v) / 2$  hebben de "hulpvariabelen"  $u$  en  $v$  hun werk gedaan en worden ze door de blokverlating weer afgevoerd: operationeel gezien houden ze op te bestaan.

Uit onze syntax volgt dat binnenblokken genest kunnen voorkomen.

Opmerking. Men is vrij in de keuze van de zogenaamde "lokale namen" (dat wil zeggen de namen die men in de aanhef van het binnenblok voor de lokale variabelen introduceert). In plaats van benoeming met  $u$  en  $v$  hadden we de lokale variabelen ook  $p$  en  $q$  mogen noemen, dat wil zeggen voor hetzelfde geld hadden we voor euclidplus mogen schrijven:

```
[[ p, q: int
  ; p:= y; q:= x
  ; do x > y → x:= x - y; q:= q + p
    || y > x → y:= y - x; p:= p + q
  od
  ; y:= (p + q)/ 2
]] .
```

Teksten die door systematische herbenoeming van lokale variabelen in elkaar kunnen worden overgevoerd zijn per definitie met elkaar equivalent.

Het is verstandig voor lokale namen geen namen te kiezen die in de omgeving van het binnenblok al betekenis hebben. (In het kielzog van ALGOL 60 staan vele programmeertalen een dergelijk dubbel gebruik wel toe; wij adviseren er geen gebruik van te maken.) (Einde van Opmerking.)

Ter vermindering van misverstand vestigen wij de aandacht op het verschil tussen een functionele specificatie en een binnenblok, in het bijzonder op het verschillende gebruik dat er van lokale variabelen wordt gemaakt. Een functionele specificatie is een uitspraak met betrekking tot de begintoestanden --dat wil zeggen waarden der lokale variabelen-- die aan de gegeven beginconditie voldoen. Aan het begin van een binnenblok hebben de daarin geïntroduceerde variabelen geen gedefinieerde waarden, en als regel zullen de eerste statements van een binnenblok aan de lokale variabelen waarden toekennen. Deze waarden zijn constanten of hangen af van de toestand van de omgeving op het moment van binnenkomst. Typisch bewerkstelligt de initialisatie van de lokale variabelen een relatie tussen lokale en globale variabelen --de laatste zijn de variabelen uit de omgeving van het binnenblok-- die vervolgens door de statements van het binnenblok in stand worden gehouden. Wij hebben dit typische gebruik van lokale variabelen bij euclidplus gezien, we zullen het nog vele malen tegenkomen.

Tot zover ons intermezzo over binnenblokken. Wij keren terug tot de repetitieve statement.

\*       \*       \*

Bij de beschrijving van het postulaat van de repetitieve statement eindigden we met "mits de repetitieve statement eindigt" en de belofte later op deze extra voorwaarde terug te komen. Nu is het moment gekomen deze belofte in te lossen.

Dat deze extra voorwaarde inderdaad gesteld moet worden, illustreert

```

| [ x: int
    {x ≥ 0}
    ; do x ≠ 0 → x := x - 1 od
    {x = 0}
] | .

```

In het bovenstaande is de beginconditie  $x \geq 0$  beslist essentieel. Mits er aan voldaan is, geeft  $x$  precies aan na hoeveel slagen de repetitie eindigt, maar voor  $x < 0$  zal de repetitie nooit eindigen: immers, hij kan slechts eindigen met  $x = 0$ , maar een negatieve  $x$  die alleen maar kan worden afgelaagd zal nooit  $= 0$  worden.

In een deterministisch programma als boven is het aantal slagen waarna de repetitie eindigt uniek door de begintoestand bepaald. Volledigheidshalve vermelden wij dat dit bij nondeterministische programma's niet het geval hoeft te zijn, zoals blijkt uit

```

| [ x: int
    {x ≥ 0}
    ; do x ≠ 0 → x := x - 1
      | x ≥ 2 → x := x - 2
    od
    {x = 0}
] | .

```

In zo'n geval bepaalt de begintoestand slechts een bovengrens voor het aantal slagen dat kan plaatsvinden.

In beide bovengegeven voorbeelden is  $x \geq 0$  kennelijk een invariant van de repetitieve statement. Tevens verlaagt elke slag  $x$  met tenminste 1. De combinatie van die twee gegevens impliceert dat het mogelijk aantal slagen van boven begrensd is (namelijk door de waarde van  $x$ ).

Bij een repetitie met invariant  $P$  zoekt men een integer functie  $vf$  van de toestand, zodat de waarde van  $vf$  als bovengrens voor het aantal geïnterpreteerd kan worden.

Wij zullen het postulaat van de repetitieve statement, maar nu met inbegrip van het beëindigingsbewijs, weer formuleren voor het geval van 2 guarded commands (de generalisatie tot meer of minder wederom aan de lezer overlatend).

De uitspraken

$$\begin{aligned} & [[ x, y, z: \text{int} \{ P \wedge B_0 \wedge vf = VF \}; S_0 \{ P \wedge vf < VF \} ]], \\ & [[ x, y, z: \text{int} \{ P \wedge B_1 \wedge vf = VF \}; S_1 \{ P \wedge vf < VF \} ]], \end{aligned}$$

en  $P \wedge (B_0 \vee B_1) \Rightarrow vf \geq 0$  in elk punt van de toestandsruimte

wettigen te zamen de uitspraak

$$\begin{aligned} & [[ x, y, z: \text{int} \\ & \quad \{ P \} \\ & \quad ; \underline{\text{do}} B_0 \rightarrow S_0 \quad \parallel \quad B_1 \rightarrow S_1 \underline{\text{od}} \\ & \quad \{ P \wedge \neg B_0 \wedge \neg B_1 \} \\ & ]]. \end{aligned}$$

Opgave. Ga na dat voor euclid en euclidplus de keuze  $vf = x + y$  aan de eisen voldoet. (Einde van Opgave.)

Opmerking. De naam "vf" is geïnspireerd door de historisch gegroeide aanduiding "variant function". We hadden ook cc mogen kiezen, want het is in wezen een soort convergentie criterium. (Einde van Opmerking.)

\* \* \*

Voorbeeld 0. Zij  $B_x$  een Boolse uitdrukking in de integer  $x$  die niet voor alle natuurlijke  $x$  false is. In dat geval kunnen wij vragen naar de minimale natuurlijke  $x$  zodat  $B_x$  geldt:

```

[[ x: int {true}
  ; linear search
    {x is de kleinste natuurlijke x zodat Bx geldt}
]] .

```

De manier om dit probleem systematisch aan te pakken is --zoals zo vaak-- een formalisering van de probleemstelling; in dit geval komt dat neer op een formalisering van de eindconditie. Het is niet voldoende te eisen dat na afloop  $Bx$  geldt, we moeten ook tot uitdrukking brengen dat het de kleinste  $x$  is, dat wil zeggen we moeten tevens eisen dat  $B$  niet geldt voor eventuele kleinere natuurlijke getallen. De analoge Boolse uitdrukking in  $i$  met  $B_i$  aanduidend, luidt de formele uitdrukking voor onze eindconditie dan ook

$$(\bigwedge i: 0 \leq i < x: \neg B_i) \wedge B_x .$$

Als --zoals hier het geval is-- de eindconditie een conjunctie is, kan men soms een geschikte invariant vinden door van de eindconditie één van de termen van de conjunctie weg te laten: de invariant moet immers een verzwakking van de eindconditie zijn. In dit geval suggereert deze methode om

P:  $(\bigwedge i: 0 \leq i < x: B_i)$

als invariant te kiezen, te meer daar de initialisatie  $x := 0$  triviaal  $P$  doet gelden.

Na deze keuze is nog maar heel weinig inventiviteit vereist. Uit de opmerking dat de waarde 0 voor  $x$  niet te groot, maar wel te klein zou kunnen zijn, concluderen we dat het programma de waarde van  $x$  moet kunnen verhogen. De inventiviteit bestaat nu uit het voorstel ons daartoe te beperken tot de assignment statement  $x := x + 1$ .

Dit leidt onmiddellijk tot de vraag onder welke beginconditie  $x := x + 1$  leidt tot een eindtoestand waarin aan  $P$  voldaan is. Directe toepassing van het postulaat van de assignment statement levert --onder inachtnaam van de definitie van  $P$ -- ons de uitspraak

$$\begin{aligned} &| [ x: \text{int} \{ (\underline{A} \ i: 0 \leq i < x + 1: \neg B i) \} \\ &\quad ; x := x + 1 \{ P \} \\ &] | \quad . \end{aligned}$$

Bovenstaande beginconditie laat zich echter herleiden:

$$\begin{aligned} &(\underline{A} \ i: 0 \leq i < x + 1: \neg B i) \\ &= \{ \text{definitie van universele quantificatie} \} \\ &(\underline{A} \ i: 0 \leq i < x: \neg B i) \wedge \neg B x \\ &= \{ \text{definitie van } P \} \\ &P \wedge \neg B x \end{aligned}$$

waaruit wij door substitutie afleiden de uitspraak

$$| [ x: \text{int} \{ P \wedge \neg B x \}; x := x + 1 \{ P \} ] | \quad .$$

Onder voorbehoud van beëindiging levert toepassing van het postulaat van de repetitieve statement ons het volgende --geannoteerde-- programma op voor linear search :

$$\begin{aligned} &\{ \text{true} \} \\ &x := 0 \{ P \} \\ &; \underline{\text{do}} \neg B x \rightarrow x := x + 1 \{ P \} \underline{\text{od}} \\ &\quad \{ P \wedge B x \} \quad . \end{aligned}$$

Op grond van de definitie van  $P$  is bovenstaande eindconditie immers juist de eindconditie uit de functionele specificatie voor linear search . Als  $X$  het gevraagde antwoord is, kan het beëindigingsargument gevoerd worden met behulp van  $vf = X - x$  . (Versterk hiervoor de invariant met de neveneis  $0 \leq x \leq X$  .)

Noot. Merk op, dat er niet het minste bezwaar tegen bestaat in

invariant of beëindigingsargument het eindantwoord te betrekken.  
(Einde van Noot.)

De moraal van het bovenstaande is dat een onderzoek in *opklimmende* volgorde ons een *minimum* waarde levert die aan een of andere voorwaarde voldoet; omgekeerd levert een onderzoek in *afnemende* volgorde ons een *maximum* waarde. (Dit alles natuurlijk in de veronderstelling dat de gezochte waarde inderdaad bestaat.) De moraal is niet diep, maar zo frequent toepasbaar dat hij een naam verdient: hij staat bekend als "The Linear Search Theorem". (Einde van Voorbeeld 0.)

Voorbeeld 1. Beschouw de functionele specificatie van het programma square root, dat voor natuurlijke  $N$  de naar beneden afgeronde vierkantswortel bepaalt

```

[[ N, a: int
   {N = N'  ∧  N ≥ 0}
   ; square root
   {N = N'  ∧  a2 ≤ N  ∧  (a + 1)2 > N}
]]

```

Een afkorting voor functionele specificaties. Bovenstaande functionele specificatie maakt duidelijk --blijkens zijn eerste regel-- dat square root zich afspeelt in een toestandsruimte opgespannen door twee integer variabelen  $N$  en  $a$ . Begin- en eindcondities drukken door middel van de term  $N = N'$  echter uit dat van die twee  $N$  helemaal niet zo variabel is: square root dient  $a^2 \leq N \wedge (a + 1)^2 > N$  te bewerkstelligen onder de bijvoorwaarde dat de waarde van  $N$  onverlet blijft. (Hierdoor wordt een quasi-oplossing als " $a := 1; N := 1$ ", die wel  $a^2 \leq N \wedge (a + 1)^2 > N$  bewerkstelligt, verworpen.)

Om niet de naam  $N'$  te hoeven introduceren en niet bij onze uitspraken steeds de weinig interessante term  $N = N'$  mee te

hoeven zeulen, voeren we de conventie in dat bij functionele specificaties, waarin de toestandsruimte met blok en binnenblok gedefinieerd wordt, de variabelen geïntroduceerd in het buitenblok in het binnenblok als constant beschouwd dienen te worden. Volgens die conventie mag de functionele specificatie van square root dan als volgt luiden:

```

| [ N: int
  ; | [ a: int
      {  $N \geq 0$  }
      ; square root
      {  $a^2 \leq N \wedge (a + 1)^2 > N$  }
  ] |
]|

```

en in alle condities hoeft de term  $N = N'$  niet meer expliciet vermeld te worden. (Einde van Een afkorting voor functionele specificaties.)

Omdat uit de eindconditie en de monotoniciteit van het kwadraat volgt dat de gewenste eindwaarde van  $a$  het kleinste natuurlijke getal is, waarvoor  $(a + 1)^2 > N$  geldt, kunnen we onmiddellijk "The Linear Search Theorem" toepassen en concluderen dat

$a := 0$ ; do  $(a + 1) * (a + 1) \leq N \rightarrow a := a + 1$  od

aan de specificatie van square root voldoet.

Opgave. Bewijs de correctheid van deze laatste oplossing direct, dat wil zeggen zonder van The Linear Search Theorem gebruik te maken. Hint: Kies als invariant  $a^2 \leq N$ . (Einde van Opgave.)

Voor grote waarden van  $N$  en dus grote uiteindelijke waarden van  $a$  is bovenstaande oplossing voor square root evenwel relatief tijdrovend doordat  $a$  per slag maar met 1 wordt opgehoogd. Uit overwegingen van efficiency is het daarom interessant uit te

zien naar een oplossing waarin  $a$  met grotere stappen stijgen kan.

Als invariant kiezen we

$$P: \quad a^2 \leq N \wedge b^2 > N \wedge 0 \leq a < b$$

waarvan de eerste twee termen verkregen zijn door in de opgegeven eindconditie de uitdrukking  $a + 1$  door een nieuwe variabele  $b$  te vervangen. Hierdoor heeft  $P$  de interessante eigenschap dat  $P \wedge b = a + 1$  de opgegeven eindconditie impliceert, zodat ons doel bereikt kan worden met een repetitieve statement met  $P$  als invariant en  $b \neq a + 1$  als guard. (De laatste term geeft een verdere beperking van het waardebereik: we zijn kennelijk niet geïnteresseerd in negatieve waarden van  $a$  en het is kennelijk de bedoeling dat  $a$  het antwoord van beneden en  $b$  het antwoord van boven benadert. Bij de invoering van nieuwe variabelen is het altijd verstandig evidente grenzen expliciet te vermelden.) Voor square root wordt nu de structuur gesuggereerd:

```

[[ b: int {N ≥ 0}
  ; bewerkstellig P
  ; bewerkstellig b = a + 1 onder invariantie van P
    {P ∧ b = a + 1, dus}
]] {a2 ≤ N ∧ (a + 1)2 > N} .

```

Voor "bewerkstellig  $P$ " zouden we kunnen kiezen

$a := 0; b := 1; \underline{\text{do}} \ b * b \leq N \rightarrow b := b + 1 \ \underline{\text{od}}$

maar dan zou "bewerkstellig  $P$ " net zo lang duren als onze om efficiency redenen verworpen eerste oplossing! Kortom:  $b$  moet veel harder omhoog, maar we mogen --zie  $P$ -- best toestaan dat  $b$  flink over het antwoord heenschiet. In plaats van met 1 verhogen, gaan we verdubbelen:

$a := 0; b := 1; \underline{\text{do}} \ b * b \leq N \rightarrow b := 2 * b \ \underline{\text{od}} \ .$

Hier wordt niet alleen  $b$  verdubbeld, maar, omdat  $a = 0$  nog geldt, ook het verschil  $b - a$ ; in "bewerkstellig  $b = a + 1$  onder invariantie van  $P$ " wordt dit verschil evenveel malen gehalveerd. Dit leidt tot de volgende oplossing voor square root --een en ander met vrij volledige annotatie-- :

```

| [ b: int {0 ≤ N}
  ; a := 0; b := 1 {a2 ≤ N ∧ 0 ≤ a < b}
  ; do b * b ≤ N → b := 2 * b {a2 ≤ N ∧ 0 ≤ a < b} od {P}
  ; do b ≠ a + 1 →
    | [ c: int {P ∧ b ≠ a + 1}
      ; c := (a + b) div 2 {P ∧ a < c < b}
      ; if c * c ≤ N →
        {c2 ≤ N ∧ b2 > N ∧ 0 ≤ c < b , d.w.z. Pac}
        a := c {P}
      | c * c > N →
        {a2 ≤ N ∧ c2 > N ∧ 0 ≤ a < c , d.w.z. Pbc}
        b := c {P}
      fi {P}
    ] | {P}
  od {P ∧ b = a + 1, dus}
] | {a2 ≤ N ∧ (a + 1)2 > N} .

```

### Opgaven.

Verifieer de bovenstaande annotatie.

Vind voor de eerste repetitieve statement een beëindigingsargument.

Vind voor de tweede repetitieve statement een beëindigingsargument dat ook valide zou zijn indien de guards van de alternatieve statement beide door `true` zouden zijn vervangen.

Laat zien dat  $P$  versterkt mag worden tot

$$P \wedge (\exists i: i \geq 0: (b - a) = 2^i) \quad .$$

Laat zien dat daarom " $c := (a + b) \text{div } 2$ " mag worden vervangen door " $c := (a + b) / 2$ ". (Einde van Opgaven.)

Was in ons eerste programma --The Linear Search-- het benodigd aantal slagen gelijk aan  $\sqrt{N}$ , in ons nieuwe programma is dat aantal evenredig met  $\log N$ .

Opmerking. Het nieuwe programma is een aanpassing van de algoritme voor het vaststellen of een gegeven waarde in een gesorteerde lijst voorkomt, die bekend staat onder de naam "Binary Search" en die later behandeld zal worden. (Einde van Opmerking.)

(Einde van Voorbeeld 1.)

Voorbeeld 2. Beschouw de functionele specificatie

```

[[ X, Y: int
  ; [[ z: int {Y ≥ 0}
    ; exponentiation {z = XY, zie Noot}
  ]I
.II .

```

Noot. Wij spreken af dat  $X^0 = 1$  geldt voor alle waarden van  $X$ , dus ook voor  $X = 0$ . (Einde van Noot.)

De taak van exponentiation is blijkens zijn functionele specificatie om een gegeven getal  $X$  tot een natuurlijke macht  $Y$  te verheffen. Onze oplossingen zijn gebaseerd op een repetitieve statement met een invariant  $P$  van de vorm

$P: \quad z \cdot h = X^Y$

die de plezierige eigenschap heeft dat we uit

$$P \wedge h = 1$$

mogen concluderen dat de eindconditie geldt.

Opmerking. Hadden we de eindconditie als

$$z \cdot 1 = X^Y$$

geschreven, dan was ook dit maal de invariant verkregen door in de eindconditie een subexpressie --hier de constante 1-- te vervan-

gen door een nieuwe variabele. Het vervangen van een dergelijke, bij wijze van spreken "onzichtbare" constante komt vaker voor. (Einde van Opmerking.)

De initiële geldigheid van  $P$  zou kunnen worden bewerkstelligd door de assignment statements

$$z := 1; h := X^Y,$$

ware het niet, dat in deze vorm de initialisatie van de hulpgrootheid  $h$  de oplossing van het oorspronkelijke probleem vergt. Omdat  $h$  maar een hulpgrootheid is, mogen we zijn waarde ook anders representeren, en we kiezen voor  $h$  een representatie, die dit initialiseringsprobleem ondervangt.

$$h \equiv X^Y$$

Volgens deze conventie wordt de waarde van  $h$  gerepresenteerd met behulp van 1 lokale variabele  $y$ . Na substitutie luidt de invariant --aangevuld met begrenzing voor  $y$  --

$$P: \quad z \cdot X^y = X^Y \wedge y \geq 0;$$

de aanvankelijke geldigheid van  $P$  kan door  $z := 1; y := Y$  worden bewerkstelligd, en aangezien  $y = 0 \Rightarrow h = 1$ , is het voldoende wanneer de dan volgende repetitieve statement onder invariantie van  $P$   $y = 0$  bewerkstelligt. Een en ander leidt tot de volgende oplossing voor exponentiation :

```

[[ y: int {Y ≥ 0}
; z:= 1; y:= Y {P}
; do y ≠ 0 → z:= z * X; y:= y - 1 {P} od
  {P ∧ y = 0, dus}
]] {z = XY}.

```

Opgave. Verifieer in het bovenstaande programma de annotatie en formuleer het beëindigingsargument. (Einde van Opgave.)

Bovenstaand programma vergt een aantal slagen gelijk aan  $Y$ , de beginwaarde van  $y$  die per slag met 1 verminderd wordt totdat  $y = 0$  geldt. Maar we kennen effectievere manieren om een variabele "klein te krijgen", zoals halveren. Voor dit doel wordt een tweede variabele voor de representatie van  $h$  geïntroduceerd.

$$h = x^y$$

Volgens deze conventie wordt de waarde van  $h$  gerepresenteerd met behulp van 2 lokale variabelen  $x$  en  $y$ . Na substitutie luidt de invariant --weer aangevuld met begrenzing voor  $y$ --

$$P: \quad z \cdot x^y = X^Y \quad \wedge \quad y \geq 0 \quad ;$$

de aanvankelijke geldigheid van  $P$  kan nu worden bewerkstelligd door  $z := 1; x := X; y := Y$ ; weer is de opgave om  $y = 0$  onder invariantie van  $P$  te bereiken. Voor even  $y$  --dat wil zeggen  $y \bmod 2 = 0$  -- is halvering van  $y$  in het algemeen een drastischer manier van verlaging dan  $y := y - 1$ ; de invariantie van  $P$  is gewaarborgd wanneer halvering van  $y$  gepaard gaat met quadratering van  $x$ .

Door toevoeging van de  $P$  niet verstorende guarded command

$$y \neq 0 \quad \wedge \quad y \bmod 2 = 0 \rightarrow x := x * x; y := y / 2$$

krijgen we in eerste instantie voor exponentiation

```

| [ x, y: int {Y ≥ 0}
  ; z := 1; x := X; y := Y {P}
  ; do y ≠ 0 → z := x * z; y := y - 1 {P}
    | y ≠ 0 ∧ y mod 2 = 0 → x := x * x; y := y / 2 {P}
    od {P ∧ y = 0, dus}
| ] {z = XY} .

```

Voor grote waarden van  $Y$  zou dit programma veel efficiënter kunnen werken, namelijk wanneer zo mogelijk altijd de tweede

guarded command voor uitvoering zou worden geselecteerd. In bovenstaand nondeterministisch programma is deze winst evenwel niet gegarandeerd, omdat altijd de eerste guarded command zou kunnen worden geselecteerd. De winst kan evenwel worden afgedwongen door de eerste guard te versterken en hem te conjugeren met de negatie van de tweede, zodat de eerste mogelijkheid alleen gekozen wordt als de tweede niet van toepassing is. Aangezien --ga dit na! --

$$(y \neq 0 \wedge \neg(y \neq 0 \wedge y \bmod 2 = 0)) \equiv (y \bmod 2 \neq 0)$$

leidt deze uitbanning van het nondeterminisme --ditmaal zonder annotatie-- tot

```

| [ x, y: int; z:= 1; x:= X; y:= Y
  ; do y mod 2  $\neq$  0  $\rightarrow$  z:= x * z; y:= y - 1
    | y  $\neq$  0  $\wedge$  y mod 2 = 0  $\rightarrow$  x:= x * x; y:= y / 2
    od
  ] | .

```

Men kan nog op allerlei manieren proberen om bovenstaande algoritme voor de machtsverheffing te "versnellen". Zo is de berekening van het product  $x * z$  de eerste keer niet nodig omdat  $z = 1$  en het product dus  $= x$  is. Zo zou men kunnen proberen te exploiteren dat onmiddellijk na  $y := y - 1$  kennelijk  $y \bmod 2 = 0$  geldt. Al zulk soort pogingen maken de programmatekst waarschijnlijk minder overzichtelijk en in elk geval een stuk langer. Bovendien zijn dat soort marginale versnellingen natuurlijk kruimelwerk vergeleken bij de winst die we boekten bij de overgang van een lineaire naar een logarithmische algoritme.

Er was een tijd dat, koste wat het kost, het snelste programma construeren gold als de hoogste wijsheid. Dit is nu gelukkig achterhaald: wij weten nu dat de Wet van de Verminderende Meer-opbrengst zich begon te wreken, dat de prijs van complexiteit niet

hoog genoeg kan worden geschat, dat compacte elegance veel belangrijker is en dat we er veel verstandiger aan doen de kruimeltjes te laten voor wat ze zijn. De keus van de goede algoritme zet veel meer zoden aan de dijk. De elegance van zijn programma's onderscheidt de vakman van de dilettant.

(Einde van Voorbeeld 2.)

## Het array

Aan onze programmanotatie ontbreekt nog iets heel wezenlijks. De introductie van de repetitieve statement was gemotiveerd door de overweging dat een korte programmatext tot een willekeurig lang durend rekenproces aanleiding moet kunnen geven. Eenzelfde soort overweging verlangt dat een korte programmatext een willekeurige hoeveelheid gegevens kan bewerken. Voor ons betekent dit, dat dezelfde programmatext betrekking moet kunnen hebben op een toestandruimte van een willekeurig --maar om praktische redenen wel eindig-- aantal dimensies. Zolang wij --zoals tot nog toe-- de individuele coördinaten van de toestandruimte stuk voor stuk in declaraties moeten opsommen, kunnen wij nooit aan dit verlangen voldoen. Wij zullen daarom onze programmanotatie de minimum uitbreiding geven die ons in staat stelt in een enkele declaratie een willekeurig aantal integers of booleans te introduceren. Wij zullen één en ander eerst aan een voorbeeld toelichten.

Zo behelst de aanhef van een functionele specificatie

```
[ N: int {N ≥ 1}
; f(i: 0 ≤ i < N): array of int
; ]
```

dat de weggelaten rest ervan begrepen moet worden in een constante omgeving bestaande uit de positieve integer  $N$  en een rijtje van  $N$  integers, respectievelijk  $f(0)$ ,  $f(1)$ , ...,  $f(N - 1)$  genaamd.

Opmerking. In bovenstaande uitleg komt de naam  $i$ , die voorkomt in de declaratie van  $f$ , niet meer voor. Deze  $i$  in de programmatext is dan ook een dummy: de tweede regel had voor hetzelfde geld

;  $f(x: 0 \leq x < N)$ : array of int

mogen luiden.

Omdat wij ons in declaraties van arrays zullen beperken tot dummies van type integer, spreken wij af in declaraties van arrays het type van de dummies *niet* expliciet te vermelden. (Einde van Opmerking.)

De integers  $f(0)$ ,  $f(1)$ , ...,  $f(N - 1)$  heten "de elementen van het array  $f$ " en  $i$  heet --voor  $0 \leq i < N$ -- de "index" of "subscript" van element  $f(i)$ . Voor array-elementen in onze programmatext luidt de syntax

$\langle \text{array element} \rangle ::= \langle \text{naam} \rangle [ \langle \text{integer expressie} \rangle ]$  ,

dat wil zeggen de naam van het array gevolgd door een tussen haakjes geplaatste integer expressie ter definitie van de bedoelde indexwaarde.

Om te laten zien hoe we met arrays en hun elementen kunnen werken voltooien we de bovengegeven aanhef:

```

| [ N: int {N ≥ 1}
  ; f(i: 0 ≤ i < N): array of int
  ; | [ s: int
    ; summation
      {s = (S i: 0 ≤ i < N: f(i))}
    ] |
  ] | ,

```

dat wil zeggen in de eindtoestand zij de waarde van  $s$  gelijk aan de som van de elementen van  $f$ . Voor onze invariant  $P$  kiezen we

$P: \quad s = (\underline{S} \ i: 0 \leq i < n: f(i)) \wedge 0 \leq n \leq N$  ,

verkregen door op voor de hand liggende wijze in de eindconditie de constante  $N$  door een nieuwe variabele, zeg  $n$ , te vervangen en zijn waardebereik op even voor de hand liggende wijze te begrenzen. Een en ander leidt tot de volgende oplossing voor summation

```

| [ n: int
  ; s := 0; n := 0
  ; do n ≠ N → s := s + f(n); n := n + 1 od
] | .

```

Opgave. Voorzie bovenstaande oplossing voor summation van annotatie en toon aan dat deze annotatie correct is en toereikend om te bewijzen dat hij inderdaad aan de functionele specificatie van summation voldoet. (Einde van Opgave.)

Opmerking. Een functionele specificatie wordt meereisend --of "sterker"-- als we de beginconditie verzwakken. Zo kunnen we de functionele specificatie van summation versterken door de beginconditie tot  $N \geq 0$  te verzwakken. Ga na, dat bovenstaande oplossing ook aan deze sterkere specificatie voldoet. Welke oplossing voldoet wel aan de oorspronkelijke specificatie van summation maar niet aan de versterkte? (Einde van Opmerking.)

In bovenstaande oplossing voor summation hebben we stiekem gebruik gemaakt van een nog niet gegeven uitbreiding van de syntax van integer expressies . We breiden hiertoe de syntax voor een intfactor uit tot

```
< intfactor >
    ::= < natuurlijk getal >
        | < naam >
        | < array element >
        | ( < integer expressie > ) .
```

In het bovenstaande voorbeeld hebben we met "array of int" een rijtje integers geïntroduceerd. Geheel analoog kunnen we met "array of bool" een rijtje booleans introduceren. De passende uitbreiding van de syntax voor Boolse expressies geschiedt door een nieuwe verschijningsvorm toe te staan voor de Boolse primary :

```
< Boolse primary >
    ::= true
        | false
        | < naam >
        | < array element >
        | < integer expressie > < relop > < integer expressie >
        | ( < Boolse expressie > ) .

        *      *      *
```

Als volgend voorbeeld beschouwen we de functionele specificatie van maxlocation

```
[[ N: int; f(i: 0 ≤ i < N): array of int {N ≥ 1}
  ; [[ k: int
      ; maxlocation
        {0 ≤ k < N ∧ (∃ i: 0 ≤ i < N: f(i) ≤ f(k))}
  ]]
]] .
```

Merk op dat de eindconditie, gezien als vergelijking in  $k$ , vanwege  $N \geq 1$  altijd oplosbaar is, maar dat deze oplossing niet uniek hoeft te zijn. Wij kunnen dus verwachten voor `maxlocation` een nondeterministisch programma te vinden.

Opgave. Geef een voldoende voorwaarde opdat de eindwaarde van  $k$  wel uniek bepaald zij. (Einde van Opgave.)

Voor onze invariant  $P$  kiezen we --als in ons vorige voorbeeld-- na introductie van een nieuwe variabele  $n$

$P: \quad 0 \leq k < n \wedge (\bigwedge i: 0 \leq i < n: f(i) \leq f(k)) \wedge 1 \leq n \leq N$

die op grond van zijn constructie weer de conclusie wettigt dat aan de eindconditie is voldaan indien  $P \wedge n = N$  geldt.

Omdat --ga dit na!--

$$P \wedge n \neq N \wedge f(n) \leq f(k) \Rightarrow P_{n+1}^n,$$

wettigt het postulaat van de assignment statement de uitspraak

```

[[ N, k, n: int; f(i: 0 ≤ i < N): array of int
  {P ∧ n ≠ N ∧ f(n) ≤ f(k)}
  ; n := n + 1
  {P}
]] .

```

Bovenstaande uitspraak vertelt ons onder welke nevenvoorwaarde de verhoging  $n := n + 1$  de geldigheid van  $P$  niet verstoort. Door eventuele aanpassing van  $k$  kunnen wij zorgen dat ook aan deze nevenvoorwaarde voldaan is. Er geldt namelijk --dankzij het postulaat van de assignment statement en de transitiviteit van de relatie  $\leq$  --

```

| [ N, k, n: int; f(i: 0 ≤ i < N): array of int
    { P ∧ n ≠ N ∧ f(n) ≥ f(k) }
    ; k := n
    { Pn+1n }
] | .

```

Opgave. Bewijs bovenstaande uitspraak. (Einde van Opgave.)

Combinatie van bovenstaande observaties levert voor max-location de oplossing

```

| [ n: int
    ; k := 0; n := 1
    ; do n ≠ N → if f(n) ≤ f(k) → skip
                  f(n) ≥ f(k) → k := n
                  fi; n := n + 1
    od
] | .

```

Opgave. Voorzie bovenstaande oplossing van de nodige annotatie en voltooi het correctheidsbewijs. (Einde van Opgave.)

\*       \*       \*

In beide bovenstaande voorbeelden was het array  $f$  constant en had het programma slechts de taak de waarde van een variabele van het type integer passend te bepalen --in het eerste voorbeeld de waarde van  $s$ , in het tweede voorbeeld de waarde van  $k$  --. We gaan nu een voorbeeld behandelen waarin het programma de waarde van het array moet berekenen, dat wil zeggen zonnodig moet aanpassen opdat aan de eindconditie voldaan zij. Dit is programma *upsort*, waarvan de functionele specificatie als volgt luidt:

```

[[ N: int {N ≥ 1}
; |[ f(i: 0 ≤ i < N): array of int
    {(B i: 0 ≤ i < N: f(i)) = X}
; upsort
    {(B i: 0 ≤ i < N: f(i)) = X ∧
    (A i, j: 0 ≤ i < j ∧ 1 ≤ j < N: f(i) ≤ f(j))}
]|
]| .

```

Laat ons eerst deze functionele specificatie zorgvuldig analyseren. Voor een gegeven array  $f(i: 0 \leq i < N)$  duidt  $(B i: 0 \leq i < N: f(i))$  de bag aan die men krijgt door de  $N$  getallen uit de rij  $f(i: 0 \leq i < N)$  in een bag te vergaren; de constante  $X$  uit begin- en eindconditie staat dus voor een bag waar  $N$  integers in zitten. Beginconditie en eerste regel van de eindconditie brengen dus tot uitdrukking dat vergaring van de getallen uit de rij  $f(i: 0 \leq i < N)$  voor en na uitvoering van upsort dezelfde bag oplevert, met andere woorden dat de uitvoering van upsort ten aanzien van de rij getallen  $f(i: 0 \leq i < N)$  beperkt is tot een permutatie --dat wil zeggen verandering van de volgorde-- van deze getallen. De tweede regel legt formeel vast, waaraan de uiteindelijke volgorde dient te voldoen: uiteindelijk dienen de getallen in de rij  $f(i: 0 \leq i < N)$  daarin in opklimmende volgorde voor te komen. (Onderling gelijke getallen in het rijtje niet uitgesloten zijnde, is  $f$  na afloop zogenaamd "ascending"; slechts als ze alle onderling verschillen is het resultaat zogenaamd "increasing".)

We verkrijgen invariant  $P$  van onze oplossing voor upsort door in de eindconditie de constante  $1$  door een nieuwe variabele, zeg  $q$ , te vervangen:

```

P:    (B i: 0 ≤ i < N: f(i)) = X ∧ 1 ≤ q ≤ N ∧
      (A i, j: 0 ≤ i < j ∧ q ≤ j < N: f(i) ≤ f(j))

```

zodat we uit  $P \wedge q = 1$  mogen concluderen, dat aan de eindconditie is voldaan.

Opmerking. Onze keuze van  $P$  lijkt erg willekeurig. Dat is hij ook. Er zijn vele andere keuzen voor  $P$  mogelijk; zij leiden tot ten minste zoveel andere oplossingen voor upsort : de ontwikkeling van sorteerrouines is een veelbeploegde akker. (Einde van Opmerking.)

Het loont de moeite om, voordat wij verder gaan, zorgvuldig te analyseren wat vooral de tweede regel van  $P$  precies behelst. Vanwege  $q \leq j < N$  staat  $f(j)$  voor een van de laatste  $N - q$  getallen uit de rij  $f(i: 0 \leq i < N)$ ; vanwege  $0 \leq i < j$  behelst  $P$  dat geen van deze laatste  $N - q$  getallen ergens door een grotere wordt voorafgegaan. Met andere woorden: de deelrij  $f(j: q \leq j < N)$  heeft zijn definitieve waarde, zodat het vervolg van de berekening zich tot een eventuele permutatie van  $f(i: 0 \leq i < q)$  kan beperken.

De voor de hand liggende structuur voor upsort is nu

```

| [ q: int {N ≥ 1}; q := N {P}
  ; do q ≠ 1 → "q := q - 1 onder invariantie van P" od
| ] .

```

Vanwege het postulaat van de assignment roept deze structuur interesse op in  $P_{q-1}^q$ . Nu geldt  $R \Rightarrow P_{q-1}^q$  met

$R: \quad P \wedge q \neq 1 \wedge (\underline{A} i: 0 \leq i < q: f(i) \leq f(q - 1))$  .

Opgave. Ga deze uitspraak zorgvuldig na. (Einde van Opgave.)

Om de twee eerste termen van  $R$  hoeven we ons niet zo te bekommeren: de eerste is de invariant en de tweede is de guard. Het is de laatste term, die onze aandacht vergt: hij behelst dat

$f(q - 1)$  , dat wil zeggen het laatste getal van de deelrij  
 $f(i: 0 \leq i < q)$  , niet wordt voorafgegaan door een groter getal.

Dit is het moment om ons de functionele specificatie van `maxlocation` te herinneren: de eindconditie van `maxlocation` drukt uit (dat  $k$  een zodanige waarde heeft) dat  $f(k)$  een maximale waarde uit het rijtje is. Het enige verschil is dat we, in plaats van in het rijtje  $f(i: 0 \leq i < N)$  waarvoor `maxlocation` geformuleerd is, nu in de plaats van een maximum van het rijtje  $f(i: 0 \leq i < q)$  zijn geïnteresseerd; voor ons doel passen wij `maxlocation` aan door over te gaan op een versie waarin  $N$  door  $q$  is vervangen. Gebruik van een aldus aangepaste versie van `maxlocation` stelt ons dus in staat om

$$(\underline{A} \ i: 0 \leq i < q: f(i) \leq f(k))$$

te bewerkstelligen; maar ons doel is de laatste term van  $R$

$$(\underline{A} \ i: 0 \leq i < q: f(i) \leq f(q - 1))$$

door verwisseling van elementen van het rijtje  $f(i: 0 \leq i < q)$  te bewerkstelligen: de "nieuwe"  $f(q - 1)$  worde gelijkgesteld aan de "oude"  $f(k)$  . In het rijtje  $f(i: 0 \leq i < q)$  dienen de waarden met index  $k$  , respectievelijk  $q - 1$  verwisseld te worden.

Verwisseling van twee elementen van een array is een zoveel voorkomende operatie dat we er een speciale statement voor introduceren, genaamd de "swap". Uitvoering van

`f:swap(k, q - 1)`

voert de in ons geval gewenste verwisseling uit; als  $k = q - 1$  --dat wil zeggen de verwisseling van een element met zichzelf-- is de uitvoering van bovengegeven `swap` verder equivalent met een `skip` . Voordat we verder op deze operatie ingaan, zullen we eerst onze behandeling van `upsort` voltooien.

```

[[ q: int {N ≥ 1}; q:= N {P}
  ; do q ≠ 1 →
    [[ k: int
      ; [[ n: int; k:= 0; n:= 1
        ; do n ≠ q → if f(n) ≤ f(k) → skip
          [] f(n) ≥ f(k) → k:= n
        fi; n:= n + 1
      od
    ] ]
    ; f:=swap(k, q - 1) {R}
  ] ] ; q:= q - 1
od
] ] .

```

\*   \*   \*

Voor de formele definitie van de semantiek van de swap --die een voorbeeld is van een zogenaamde "array modifier"-- hebben we een nieuwe notatie nodig. Ter inleiding het volgende.

De waarde van een variabele van het type "array of ..." is een functie gedefinieerd op een eindig aantal integers, die te zamen het zogenaamde "domein" van de functie vormen. In formule wordt "f en g zijn dezelfde functie" uitgedrukt door

(A i:: f(i) = g(i))

waarbij f(i) = g(i) geacht wordt te gelden wanneer i buiten beider domein valt --zoiets als "ongedefinieerd = ongedefinieerd"-- maar niet wanneer i binnen het ene en buiten het andere domein valt --zoiets als "gedefinieerd ≠ ongedefinieerd"-- ; met die conventie hebben we gevangen dat voor gelijkheid functies ten minste hetzelfde domein moeten hebben.

Onze interesse gaat nu uit naar een functie g , die maar heel

weinig van  $f$  afwijkt, bijvoorbeeld slechts mogelijk in punt  $h$ , precieser we beschouwen tussen  $f$  en  $g$  (en  $h$  en  $p$ ) de relatie

$$g(h) = p \wedge (\bigwedge i: i \neq h: f(i) = g(i)) \quad .$$

De functie  $g$  die aan bovenstaande betrekking voldoet wordt genoteerd als  $(f; h: p)$ . Als  $h$  behoort tot het domein van  $f$ , heeft  $(f; h: p)$  hetzelfde domein, anders is het domein van  $(f; h: p)$  het domein van  $f$ , uitgebreid met het punt  $h$ .

Uit bovenstaande definitie volgen de rekenregels

$$\begin{aligned} (f; h: p)(i) &= p \quad \text{als } i = h \\ (f; h: p)(i) &= f(i) \quad \text{als } i \neq h \quad . \end{aligned}$$

Opmerking. De lezer die schrikt bij het idee van een expressie --zoals  $(f; h: p)$ -- die een functie voorstelt, realiseer zich dat het idee uit de differentiaalrekening heel vertrouwd is. (Einde van Opmerking.)

Het gebruik van  $(f; h: p)$  wordt nog even uitgesteld; in verband met de swap gaat onze onmiddellijke interesse meer uit naar een functie  $g$  die in twee punten van het domein van  $f$  verschillen kan, dat wil zeggen die (voor zekere  $h$ ,  $k$ ,  $p$  en  $q$ ) voldoet aan

$$g(h) = p \wedge g(k) = q \wedge (\bigwedge i: i \neq h \wedge i \neq k: f(i) = g(i)) \quad .$$

Opmerking. Voor  $h = k \wedge p \neq q$  bestaat zo'n  $g$  niet, voor  $h \neq k \vee p = q$  bestaat zo'n  $g$  altijd. (Einde van Opmerking.)

Onder de veronderstelling  $h \neq k \vee p = q$  wordt de functie  $g$  die aan bovenstaande betrekking voldoet genoteerd als  $(f; h, k: p, q)$ . Uit bovenstaande definitie volgen de rekenregels

```

(f; h, k: p, q)(i) = p    als i = h
(f; h, k: p, q)(i) = q    als i = k
(f; h, k: p, q)(i) = f(i) als i ≠ h ∧ i ≠ k .

```

En nu hebben we het --zij het wat moeizame-- gereedschap om de semantiek van de swap te definiëren. Voor een variabele  $f$  van het type "array of ..." is de semantiek van  $f:\text{swap}(h, k)$  gelijk die van de overigens niet in programma's toegelaten assignment statement

```
f := (f; h, k: f(k), f(h)) .
```

Toepassing van het postulaat van de assignment levert de structuur van de algemene uitspraak over de swap :

```

| [ h, k: int; f(.....): array of ...
  {Rf
   {f; h, k: f(k), f(h)}}
  ; f:swap(h, k)
  {R}
] | .

```

Opgave. Toon de juistheid aan van

```

| [ h, k: int; f(.....): array of ...
  {R ∧ f(h) = f(k)}
  ; f:swap(h, k)
  {R ∧ f(h) = f(k)}
] | voor willekeurige R .

```

Met andere woorden onder de nevenvoorwaarde  $f(h) = f(k)$  functioneert  $f:\text{swap}(h, k)$  als een skip ; een gevolg van deze stelling is dat zulks ook het geval is onder de nevenvoorwaarde  $h = k$  .  
(Einde van Opgave.)

Opgave. Toon voor willekeurige  $R$  de juistheid aan van

```

[[ h, k: int; f(.....): array of ...
  {R}
  ; f:swap(h, k); f:swap(h, k)
  {R}
]] .

```

Met andere woorden de operatie `f:swap(h, k)` is zijn eigen inverse. (Einde van Opgave.)

Opmerking. Met `a` en `b` variabelen van hetzelfde type "array of ..." zou er geen enkel logisch bezwaar tegen zijn om voor arrays een assignment statement van de vorm `a:= b` in onze programma's toe te laten; uit een oogpunt van uniformiteit is het zelfs aantrekkelijk. Om twee praktische redenen zullen wij de algemene assignment aan de array-variabele evenwel *niet* introduceren.

De eerste praktische reden is dat in zo goed als alle implementatietechnieken de uitvoering van een array-assignment `a:= b` in het algemeen veel en veel meer tijd kost dan de integer assignment `x:= y`. Altijd met een half oog naar de efficiency van onze programma's kijkend, beschouwen we `x:= y` wel als een primitivum, maar de tijdrovende `a:= b` zeker niet en het zou in dit opzicht misleidend zijn beide operaties op precies dezelfde manier te noteren.

De tweede praktische reden is dat we onmiddellijk zouden worden uitgenodigd om, net als we bij integers en Booleans hebben gedaan, toelaatbare expressies van het type "array of ..." te introduceren. Maar terwijl bij eenvoudige types, zoals integer en Boolean, het aantal zinvolle operatoren zo beperkt is, dat je ze bij wijze van spreken "allemaal" introduceert, is bij arrays het hek van de dam. Niet alleen is het daardoor erg moeilijk een verdedigbare keuze te maken --K. Iverson heeft het met het ontwerp van APL geprobeerd-- maar erger nog is dat men tevens de regels

moet ontwikkelen --en beheersen!-- volgens welke zulke expressies (speciaal in bewijsvoeringen) gemanipuleerd en vooral vereenvoudigd kunnen worden. (Het is dan ook ongebruikelijk dat van APL-programma's de correctheid wordt aangetoond.) (Einde van Opmerking.)

Het formalisme  $(f; h, k: f(k), f(h))$  ter aanduiding van het door de modifier  $f: \text{swap}(h, k)$  gevormde array is te moeizaam om veel gebruikt te worden. We hebben het gegeven om te benadrukken dat deze modificatoren beschouwd moeten worden als operatoren op het "gehele" array, en voor de volledigheid. In de praktijk beroept men zich zelden direct op dit formalisme maar in plaats daarvan op stellingen, die met behulp van dit formalisme wel bewezen kunnen worden. Wij geven een paar voorbeelden.

- (i) De statements  $f: \text{swap}(h, k)$  en  $f: \text{swap}(k, h)$  zijn semantisch equivalent.
- (ii)  $\begin{array}{l} \text{I} [ h, k, z, N: \text{int}; f(i: 0 \leq i < N): \underline{\text{array of int}} \\ \quad \{ h = H \wedge k = K \wedge z = Z \wedge N = M \wedge 0 \leq h, k < N \} \\ \quad ; f: \text{swap}(h, k) \\ \quad \{ h = H \wedge k = K \wedge z = Z \wedge N = M \wedge 0 \leq h, k < N \} \\ \text{I} \end{array}$
- (iii)  $\begin{array}{l} \text{I} [ h, k, N: \text{int}; f(i: 0 \leq i < N): \underline{\text{array of int}} \\ \quad \{ f(h) = X \wedge f(k) = Y \} \\ \quad ; f: \text{swap}(h, k) \\ \quad \{ f(k) = X \wedge f(h) = Y \} \\ \text{I} \end{array}$
- (iv)  $\begin{array}{l} \text{I} [ h, k, z, N: \text{int}; f(i: 0 \leq i < N): \underline{\text{array of int}} \\ \quad \{ z \neq h \wedge z \neq k \wedge f(z) = X \} \\ \quad ; f: \text{swap}(h, k) \\ \quad \{ z \neq h \wedge z \neq k \wedge f(z) = X \} \\ \text{I} \end{array}$

```
(v)  |[ h, k, N: int; f(i: 0 ≤ i < N): array of int
      {0 ≤ A ≤ B ≤ N ∧ A ≤ h < B ∧ A ≤ k < B ∧
       (⌊ i: A ≤ i < B: f(i)) = X}
      ; f: swap(h, k)
      {(⌊ i: A ≤ i < B: f(i)) = X}
    ]|
      enzovoorts.
      *
      *
      *
```

Als volgend voorbeeld van het gebruik van de swap zullen we een programma ontwikkelen dat voldoet aan de volgende functionele specificatie voor rotation

```
[ k, N: int {N ≥ 1 ∧ 0 ≤ k < N}
; |[ f(i: 0 ≤ i < N): array of int
  {(⌊ i: 0 ≤ i < N: f(i) = X((k + i) mod N))}
  ; rotation
  {(⌊ i: 0 ≤ i < N: f(i) = X(i))}
]|
]| .
```

Wij beginnen weer met een zorgvuldige analyse van wat deze functionele specificatie behelst. Kennelijk mogen we de rij  $X(i: 0 \leq i < N)$  identificeren met de eindwaarde van array f. Uit de beginconditie volgt dat alle elementen van rij X aanvankelijk ook in array f voorkomen, maar op een andere plaats.

Laat ons ter analyse van de beginconditie daaruit eerst de operatie "mod N" elimineren door de gevallen  $i + k < N$  te separeren van de gevallen  $i + k \geq N$ . De beginconditie wordt dan

```
(⌊ i: 0 ≤ i < N - k: f(i) = X(k + i)) ∧
(⌊ i: N - k ≤ i < N: f(i) = X(k + i - N)) .
```

Wij kunnen dit wat symmetrischer opschrijven met behulp van de constante h gegeven door

$$h + k = N$$

en door de vervanging van  $i$  in de tweede term door  $h + j$ . Wij krijgen dan

$$\begin{aligned} (\underline{A} \ i: 0 \leq i < h: f(i) = X(k + i)) \quad \wedge \\ (\underline{A} \ j: 0 \leq j < k: f(h + j) = X(j)) \end{aligned} \quad .$$

Ter verkorting noemen we de deelrij  $X(j: 0 \leq j < k)$   $K$  en de deelrij  $X(i: k \leq i < k + h)$   $H$ . Concatenatie van rijen aangegeven met  $\cdot$ , geldt dan

$$X = K \cdot H$$

en luidt de eindconditie derhalve

$$f = K \cdot H$$

en de beginconditie

$$f = H \cdot K \quad .$$

Kortom: de deelrijen  $H$  en  $K$  moeten "van plaats verwisselen", maar dat hebben we dan maar wel tussen aanhalingstekens gezet omdat de twee deelrijen niet even lang hoeven te zijn.

Voor een willekeurige rij  $R$  van eindige lengte duidt men met  $\text{rev } R$  --van "reverse"-- de rij aan met dezelfde elementen als  $R$ , doch in omgekeerde volgorde. De operatie  $\text{rev}$  is kennelijk zijn eigen inverse, dat wil zeggen

$$\text{rev}(\text{rev } R) = R \quad \text{voor elke } R \quad .$$

Belangrijker is voor ons de stelling

$$\text{rev}(H \cdot K) = (\text{rev } K) \cdot (\text{rev } H) \quad ,$$

die leidt tot de volgende oplossing voor rotation.

```

| [ x, y: int
    {f = H • K}
    ; x := 0; y := N - 1
    ; do x < y → f:swap(x, y); x := x + 1; y := y - 1 od
      {f = (rev K) • (rev H)}
    ; x := k; y := N - 1
    ; do x < y → f:swap(x, y); x := x + 1; y := y - 1 od
      {f = (rev K) • H}
    ; x := 0; y := k - 1
    ; do x < y → f:swap(x, y); x := x + 1; y := y - 1 od
      {f = K • H}
| ] .

```

Opgave. Bewijs de correctheid van het deelprogramma voor bijvoorbeeld de eerste toepassing van `rev`. In tegenstelling tot in de programmatekst moet in het bewijs waarschijnlijk wel onderscheid worden gemaakt tussen  $N$  even en  $N$  oneven. (Einde van Opgave.)

Opmerking. In de meeste programmanotaties beschikt men wel over een of ander afkortingsmechanisme zodat men niet tot tekstherhalingen als boven is gedwongen. (Einde van Opmerking.)

\*       \*       \*

Er bestaat voor `rotation` een heel andere oplossing, gebaseerd op de volgende overweging. Vanuit de begintoestand

$$f = H \cdot K$$

moet de eindtoestand

$$f = K \cdot H$$

bereikt worden. Nu beschouwen we het geval dat  $H$  ten minste zo lang als  $K$  is. In dat geval kunnen we  $H$  als  $H_0 \cdot H_1$  schrijven, waarbij  $H_0$  net zo lang is als  $K$ . Als we dat substitueren, moet dus de overgang van

$$f = H_0 \cdot H_1 \cdot K \quad \text{naar} \quad f = K \cdot H_0 \cdot H_1$$

worden bewerkstelligd. Omdat  $H_0$  en  $K$  even lang zijn is de overgang van

$$f = H_0 \cdot H_1 \cdot K \quad \text{naar} \quad f = K \cdot H_1 \cdot H_0$$

door een verwisseling van de deelrijen  $H_0$  en  $K$  makkelijk te realiseren. Dan rest de overgang van

$$f = K \cdot H_1 \cdot H_0 \quad \text{naar} \quad f = K \cdot H_0 \cdot H_1$$

maar dat is een operatie van dezelfde soort als de oorspronkelijke, maar nu op de kortere deelrij  $H_1 \cdot H_0$ . Het geval dat  $K$  ten minste zo lang is als  $H$  laat zich soortgelijk behandelen, namelijk door  $K$  op te splitsen als concatenatie van twee deelrijen, waarvan de achterste zo lang als  $H$  is. We krijgen op deze manier een algoritme waarin in elke stap òf het aantal elementen vooraan  $f$  òf het aantal elementen achteraan  $f$ , die hun definitieve waarde hebben, wordt uitgebreid.

Wij voeren vier variabelen in,  $p$  en  $q$  zodat de voorste  $p$  elementen van  $f$  en de achterste  $N - q$  elementen van  $f$  hun definitieve waarde hebben en een  $r$  en  $s$  die beschrijven hoe de tussenliggende waarden van  $f$  nog geroteerd moeten worden. De globale invariant van ons programma wordt

$$\begin{aligned} P: & 0 \leq p \leq N \wedge 0 \leq q \leq N \wedge 0 \leq r \wedge 0 \leq s \wedge p + s + r = q \wedge \\ & (\underline{A} \ i: 0 \leq i < p \vee q \leq i < N: f(i) = X(i)) \wedge \\ & (\underline{A} \ i: p \leq i < p + r: f(i) = X(i + s)) \wedge \\ & (\underline{A} \ i: q - s \leq i < q: f(i) = X(i - r)) \end{aligned}$$

Op grond van de beginconditie bewerkstelligt de initialisatie

$$p := 0; q := N; s := k; r := N - k$$

de invariant  $P$ . Verder volgt uit  $P \wedge (r = 0 \vee s = 0)$  de eindconditie.

Wij zullen nu laten zien

- 1) hoe in het geval  $0 < s \leq r$ , onder invariantie van  $P$ ,  $p$  met  $s$  verhoogd en  $r$  met  $s$  verlaagd kan worden.
- 2) hoe in het geval  $0 < r \leq s$ , onder invariantie van  $P$ ,  $q$  en  $s$  beide met  $r$  verlaagd kunnen worden.

1)  $0 \leq s \leq r$  De assignments  $p := p + s$ ;  $r := r - s$  laten de eerste regel van  $P$  onverlet. De beginconditie dat voor dit tweetal na afloop de overige regels van  $P$  gelden is --bij gratie van het postulaat van de assignment--

$$\begin{aligned} & (\underline{A} \ i: 0 \leq i < p + s \vee q \leq i < N: f(i) = X(i)) \wedge \\ & (\underline{A} \ i: p + s \leq i < p + r: f(i) = X(i + s)) \wedge \\ & (\underline{A} \ i: q - s \leq i < q: f(i) = X(i - r + s)) \end{aligned}$$

Als we in de laatste regel  $i$  vervangen door  $i + r$ , luidt hij --na enige simplificatie--

$$(\underline{A} \ i: p \leq i < p + s: f(i + r) = X(i + s))$$

en door uit de eerste regel  $-- p + s < q ! --$   $s$  termen af te splitsen vinden we voor onze beginconditie

$$\begin{aligned} & (\underline{A} \ i: 0 \leq i < p \vee q \leq i < N: f(i) = X(i)) \wedge \\ & (\underline{A} \ i: p + s \leq i < p + r: f(i) = X(i + s)) \wedge \\ & (\underline{A} \ i: p \leq i < p + s: f(i) = X(i) \wedge f(i + r) = X(i + s)) \end{aligned}$$

Door ook in  $P$  in de laatste regel  $i$  door  $i + r$  te vervangen, vinden we voor de laatste drie regels van  $P$  --  $0 < s \leq r$  --

$$\begin{aligned} & (\underline{A} \ i: 0 \leq i < p \vee q \leq i < N: f(i) = X(i)) \wedge \\ & (\underline{A} \ i: p + s \leq i < p + r: f(i) = X(i + s)) \wedge \\ & (\underline{A} \ i: p \leq i < p + s: f(i) = X(i + s) \wedge f(i + r) = X(i)) \end{aligned}$$

en na deze herleidingen zien we, dat onze beginconditie voor  $p := p + s$ ;  $r := r - s$  slechts in de laatste regel van  $P$  verschilt:

uit  $P$  kunnen we de beginconditie bewerkstelligen door voor  $i: p \leq i < p + s$   $f: \text{swap}(i, i + r)$  uit te voeren.

2)  $0 < r \leq s$ . De analyse van dit geval laten wij als oefening aan de lezer.

Na de gevraagde aanvulling leiden deze bespiegelingen tot de volgende oplossing voor rotation .

```

[[ p, q, r, s: int
  ; p:= 0; q:= N; s:= k; r:= N - k
  ; do r ≠ 0 ∧ s ≠ 0 →
    [[ i: int
      ; if s ≤ r →
        i:= p
        ; do i ≠ p + s → f:swap(i, i + r); i:= i + 1 od
        ; p:= p + s; r:= r - s
      [] r ≤ s →
        i:= q - r
        ; do i ≠ q → f:swap(i, i - s); i:= i + 1 od
        ; q:= q - r; s:= s - r
      fi
    ] ]
  od
] ] .

```

Opmerking. Concentreren wij in bovenstaande oplossing onze aandacht op  $r$  en  $s$ , dan herkennen we de algorithmen van Euclides voor de grootste gemene deler. De lezer wordt uitgenodigd zich ervan te overtuigen, dat het totale aantal malen dat  $f: \text{swap}$  wordt uitgevoerd in bovenstaande versie van rotation gelijk is aan  $N - \text{ggd}(N, k)$ . (Einde van Opmerking.)

Het zal de oplettende lezer niet zijn ontgaan, dat we de twee oplossingen voor rotation in nogal verschillende stijlen hebben ontwikkeld.

In de eerste ontwikkeling werd de rij  $f$  opgevat als concatenatie van twee deelrijen en werden de (begin-, tussen-, en eind-) condities dan ook geformuleerd door (met behulp van de functie  $rev$ ) voor de rij  $f$  een formule te geven. De individuele elementen van array  $f$  kwamen daarin niet meer ter sprake.

In de tweede ontwikkeling zijn we ook zo begonnen, maar hebben we uiteindelijk de invariant geformuleerd en de gedetailleerde analyse gepleegd in termen van de individuele elementen van het array  $f$ . We hebben de tweede ontwikkeling zo gegeven om te laten zien dat het zo kan, en dat de formele analyse keurig oplevert welke swaps uitgevoerd moeten worden. Er kleeft echter een bezwaar aan: het is niet denkbeeldig dat men door de bomen het zicht op het bos verliest.

Aan de eerste ontwikkeling kleeft dat bezwaar veel minder. Daar praten we nauwelijks over het array en zijn individuele elementen: we spreken één keer af hoe we een array een rij laten representeren en voeren dan ons betoog in termen van rijen. Deze indirectere behandeling komt de kortheid van het betoog ten goede, en geniet daarom de voorkeur. Wij moeten er echter op wijzen dat in dit voorbeeld de afspraak hoe een array een rij representeert wel heel triviaal was. Zodra arrays gebruikt worden om ingewikkeldere structuren te representeren --bomen, grafen in het algemeen, puntverzamelingen, enzovoorts-- eist de indirecte behandeling wel dat de representatie-afspraken precies en volledig wordt geformuleerd.

Wij voeren ten slotte onze tweede en laatste "array modifier"

in. Immers, alleen de swap is niet voldoende: de swap permuteert slechts waarden die al in het array voorkomen, maar stelt ons niet in staat de collectie waarden die in het array voorkomen te veranderen of uit te breiden. Zo'n soort faciliteit hebben we beslist nodig, aangezien de declaratie van een lokaal array in een binnenblok een array introduceert waar nog geen enkele waarde in voorkomt.

Wij brengen in herinnering de betekenis van de notatie  $(f; h: p)$ ; zij  $f$  een array (dat wil zeggen array of int dan wel array of bool), zij  $h$  een integer expressie en  $p$  een (integer dan wel Boolse) expressie, dan is  $(f; h: p)$  een nieuwe array-waarde gegeven door

$$\begin{aligned} (f; h: p)(i) &= p && \text{als } i = h \\ (f; h: p)(i) &= f(i) && \text{als } i \neq h \end{aligned}$$

Met andere woorden,  $(f; h: p)$  wijkt maar heel weinig van  $f$  af: gezien als functies op de natuurlijke getallen stemmen ze overal overeen met mogelijke uitzondering van punt  $h$ , waar  $(f; h: p)$  in elk geval de waarde  $p$  heeft, onafhankelijk van  $f$ , die in dat punt zelfs ongedefinieerd zou kunnen zijn.

Onze laatste array modifier heeft de semantiek van de overigens in onze programma's niet toegelaten assignment statement

$$f := (f; h: p)$$

en wordt in onze programma's geschreven als

$$f:(h) = p$$

Toepassing van het postulaat van de assignment levert de structuur van de algemene uitspraak over deze array modifier:

```

| [ f(.....): array of ...
  {Rf(f, E0: E1)}
  ; f:(E0)= E1
  {R}
| |

```

waar E0 een integer expressie is en E1 een expressie van hetzelfde type als de elementen van  $f$ .

Opmerking. Wie " $f:(E0)= E1$ " wil uitspreken, geven wij in overweging dit als " $f$  wordt in E0 gelijk E1" te doen. (Einde van Opmerking.)

Terzijde. De introductie van deze "elementary modifier" maakt de eerder geïntroduceerde swap, zij het slechts uit zuiver logisch oogpunt bezien, overbodig. Voor een integer array  $f$  heeft

```
f:swap(h, k)
```

hetzelfde effect als het binnenblok

```

| [ x: int
  ; x:= f(h); f:(h)= f(k); f:(k)= x
| | ,

```

een equivalentie waarvan de lezer zich op allerlei manieren kan overtuigen. Wij zullen aan deze equivalentie *niet* de consequentie verbinden de swap dan maar weer af te voeren. (Einde van Terzijde.)

Als voorbeeld zullen wij nu het programma laten zien dat, gegeven de decimale cijfers van twee (nonnegatieve) addenda, de decimale cijfers van de som berekent. De functionele specificatie van "decipius" luidt

```

| [ N: int {N ≥ 0}
  ; a, b(i: 0 ≤ i < N): array of int
    { (A i: 0 ≤ i < N: 0 ≤ a(i) < 10 ∧ 0 ≤ b(i) < 10) }
  ; | [ s(i: 0 ≤ i < N + 1): array of int
    ; deciplus
      { (A i: 0 ≤ i < N + 1: 0 ≤ s(i) < 10) ∧
        dec(N + 1, s) = dec(N, a) + dec(N, b) }
    ] |
  ] |

```

waarin gebruik is gemaakt van de functie `dec` die aan een cijferrij de bijbehorende waarde toevoegt waarvan die cijferrij de decimale representatie is, precieser

$$\text{dec}(n, x) = (\sum_{i: 0 \leq i < n: x(i) \cdot 10^i}) \quad .$$

Merk op dat in de functionele specificatie `array s` een element meer heeft dan arrays `a` en `b`.

Wij geven het programma en laten het opstellen van de invariant annex het leveren van het correctheidsbewijs aan de lezer over. Variabele `c` (= "carry", het Engels voor "overdracht") speelt de rol van het "1 onthouden" zoals wij allemaal van de schoolbanken kennen. (Neem in de invariant ook de grenzen voor `c` op opdat bewezen kan worden dat de elementen van `s` aan de gestelde ongelijkheden voldoen.)

Voor `deciplus` bieden wij de volgende oplossing aan:

```

| [ n, c: int
  ; n := 0; c := 0
  ; do n ≠ N →
    | [ z: int
      ; z := c + a(n) + b(n)
    ] |
  ] |

```

```

; if  $z \geq 10 \rightarrow s:(n) = z - 10; c := 1$ 
   $z < 10 \rightarrow s:(n) = z; c := 0$ 
fi ;  $n := n + 1$ 
]l
od
;  $s:(n) = c$ 
]l .

```

In de laatste regel van de eindconditie hebben we gebruik gemaakt van de functie `dec` . Deze vereenvoudigt niet alleen de formulering van de eindconditie maar ook die van de invariant en het correctheidsbewijs dat gebruik maakt van de volgende stelling over `dec` :

$$\text{dec}(n + 1, x) = \text{dec}(n, x) + x(n) \cdot 10^n .$$

(We noemden dit met een groot woord een stelling; het is eigenlijk maar een stellinkje, want het volgt onmiddellijk uit de recursieve definitie van de sommatie S .)

En passant vestigen wij er de aandacht op dat de individuele array-elementen wat op de achtergrond zijn geraakt: het betoog wordt grotendeels gevoerd in termen van de functie `dec` , gedefinieerd op (een heel stuk van) arrays.

Opgave. Met  $a(i: 0 \leq i < M)$  ,  $b(i: 0 \leq i < N)$  en  $s(i: 0 \leq i < M + N)$  laat zich het analoge probleem "decitimes" stellen van de cijfersgewijze decimale vermenigvuldiging. Stel de functionele specificatie op (waarvan de tweede regel

;  $a(i: 0 \leq i < M), b(i: 0 \leq i < N)$ : array of int

luide) en los het aldus gestelde probleem zonder de introductie van lokale arrays op onder de nevenconditie dat ook tussentijds geen van de elementen van `s` groter dan 9 wordt. (Einde van Opgave.)

Hiermee is de introductie van onze programmanotatie voltooid, zij het dat we de laatste uitbreiding van de formele syntax ten behoeve van de arrays bij wijze van oefening aan de lezer overlaten. De rest van dit collegedictaat is gewijd aan een collectie voorbeelden, waarin verschillende oplossingsstrategieën zullen worden getoond.

### De minimale segmentsom

Voor een integer array  $f(i: 0 \leq i < N)$  is voor  $0 \leq i \leq j \leq N$  de segmentsom  $Q(i, j)$  gedefinieerd door

$$Q(i, j) = (\sum_{h: i \leq h < j: f(h)) \quad .$$

(Merk op dat ook sommen van lege segmenten zijn toegestaan en dat als  $f$  leeg is --dat wil zeggen  $N = 0$  -- de (lege) segmentsom  $Q(0, 0)$  nog gedefinieerd is.) De bedoeling is om de minimale segmentsom te bepalen, precieser, een oplossing te vinden voor minsegsum, gespecificeerd door

```

| [ N: int {N ≥ 0}
  ; f(i: 0 ≤ i < N): array of int
  ; | [ x: int
    ; minsegsum
      {x = (MIN i, j: 0 ≤ i ≤ j ≤ N: Q(i, j))}
    ]
  ]
  .

```

Het aantal paren  $(i, j)$  waarover het minimum van  $Q(i, j)$  bepaald moet worden bedraagt  $(N + 1) \cdot (N + 2) / 2$ , de onafhankelijke berekening van  $Q(i, j)$  vergt voor een gegeven paar  $(i, j)$  een rekentijd evenredig met  $j - i$  en het allernaïefste programma

zou zodoende een rekentijd evenredig met  $N^3$  vergen. Maar het kan wel wat efficiënter! (Aangezien elk element van array  $f$  ten minste 1 maal in het rekenproces betrokken dient te worden, accepteren we als evident dat een rekentijd evenredig met  $N$  het beste is waar we op mogen hopen.)

Met het oog op de eindconditie kiezen we zoals gebruikelijk voor de invariant in eerste instantie  $P_0$ , gegeven door

$P_0: \quad 0 \leq n \leq N \wedge x = \{\text{MIN } i, j: 0 \leq i \leq j \leq n: Q(i, j)\} \quad .$

Deze invariant is daarom zo aantrekkelijk omdat blijkens de definitie van  $Q(i, j)$  de elementen van  $f(i: n \leq i < N)$  *niet* in  $P_0$  voorkomen. Dit heeft voor een minsegsum van de structuur

```

[[ n: int
  ; "bewerkstellig  $P_0$  voor  $n = 0$ "
  ; do  $n \neq N \rightarrow$ 
    {  $P_0 \wedge n \neq N$  }
    "aanpassing van  $x$ "
    {  $P_0^n_{n+1}$  }
    ;  $n := n + 1$  {  $P_0$  }
  od
]]
```

het prettige gevolg dat de elementen van  $f$  stuk voor stuk --en wel in volgorde van opklimmende index-- in de berekening worden betrokken.

De initialisatie is triviaal:  $x := 0; n := 0$  .

Voor "aanpassing van  $x$ " werken we  $P_0^n_{n+1}$  uit:

$0 \leq n + 1 \leq N \wedge x = \{\text{MIN } i, j: 0 \leq i \leq j \leq n + 1: Q(i, j)\} \quad .$

De eerste term is een consequentie van de beginconditie. Omdat

$$(0 \leq i \leq j \leq n + 1) \equiv (0 \leq i \leq j \leq n) \vee (0 \leq i \leq j = n + 1)$$

laat de tweede term zich herschrijven als

$$x = \min(\underbrace{\text{MIN}}_{i: 0 \leq i \leq j \leq n: Q(i, j)} , \underbrace{\text{MIN}}_{i: 0 \leq i \leq n + 1: Q(i, n + 1)}) .$$

Het eerste argument van  $\min$  kennen we uit  $P_0$  ; het tweede argument is evenwel nieuw. Het suggereert de introductie van een nieuwe variabele, bijvoorbeeld  $y$  , en een nieuwe invariant  $P_0 \wedge P_1$  met  $P_1$  gegeven door

$$P_1: \quad y = \underbrace{\text{MIN}}_{i: 0 \leq i \leq n: Q(i, n)} .$$

Mits  $P_1^n_{n+1}$  reeds geldt, kan de "aanpassing van  $x$ " geschieden door

$$x := \min(x, y) .$$

Voor de bewerkstelling van  $P_1^n_{n+1}$  werken we deze uit

$$\begin{aligned} y &= \underbrace{\text{MIN}}_{i: 0 \leq i \leq n + 1: Q(i, n + 1)} \\ &= \min(\underbrace{\text{MIN}}_{i: 0 \leq i \leq n: Q(i, n + 1)}, Q(n + 1, n + 1)) \\ &= \min(\underbrace{\text{MIN}}_{i: 0 \leq i \leq n: Q(i, n)} + f(n), 0) \\ &= \min(\underbrace{\text{MIN}}_{i: 0 \leq i \leq n: Q(i, n)} + f(n), 0) , \end{aligned}$$

waarna we in het eerste argument van  $\min$  (in de laatste regel) de uitdrukking uit  $P_1$  herkennen. Zo komen we voor  $\min\text{segsum}$  tot de structuur

```

| [ n, y: int; x:= 0; y:= 0; n:= 0 {P0 ∧ P1}
;  do n ≠ N → {n ≠ N ∧ P0 ∧ P1}
      y:= min(y + f(n), 0) {n ≠ N ∧ P0 ∧ P1nn+1}
      ; x:= min(x, y) {n ≠ N ∧ P0nn+1 ∧ P1nn+1}
      ; n:= n + 1 {P0 ∧ P1}
    od
|] .

```

De functie `min` eliminerend en de --evidente-- invariant  $x \leq 0$  exploiterend komen we tot de oplossing voor `minsegsum` (zonder annotatie)

```

[[ n, y: int; x:= 0; y:= 0; n:= 0
  ; do n  $\neq$  N  $\rightarrow$  y:= y + f(n)
      ; if y  $\geq$  0  $\rightarrow$  y:= 0
          [] y < 0  $\rightarrow$  if x  $\leq$  y  $\rightarrow$  skip
              [] x > y  $\rightarrow$  x:= y
          fi
      fi
      ; n:= n + 1
  od
]] .

```

En hiermee zijn we kennelijk aan het einde van onze behandeling van `minsegsum`, want onze oplossing vergt een rekentijd evenredig met `N` en we hebben al vastgesteld dat dat het best haalbare is. De gevolgde strategie heeft vaker tot verrassend efficiënte oplossingen geleid.

In eerste instantie voerden we alleen `P0` in die beschrijft wat --hier in de variabele `x`-- dient te beklijven opdat na afloop het gevraagde antwoord bekend is. De vereiste invariantie van `P0` stelt een nieuw probleem, dat op zijn beurt dicteert welk extra gegeven nog meer uit het verleden dient te beklijven; zo werden `P1` en `y` geïntroduceerd. Daar was het dit keer mee klaar; in ingewikkeldere problemen kan het gebeuren dat de variabelen van de lokale toestandsruimte in een groter aantal stappen worden ingevoerd.

## De coïncidentietelling

Gegeven zijn twee monotoon stijgende integer rijen  $F(i: 0 \leq i < M)$  en  $G(j: 0 \leq j < N)$ . Gevraagd het aantal waarden dat in beide rijen voorkomt. Precieser, we zoeken een oplossing voor `coincount`, gespecificeerd door

```

| [ M, N: int {M ≥ 0 ∧ N ≥ 0}
  ; F(i: 0 ≤ i < M), G(j: 0 ≤ j < N): array of int
    {(A i, j: 0 ≤ i < j < M: F(i) < F(j)) ∧
     (A i, j: 0 ≤ i < j < N: G(i) < G(j))}
  ; | k: int
    ; coincount
      {k = (N i, j: 0 ≤ i < M ∧ 0 ≤ j < N: F(i) = G(j))}
    |
  ] | .

```

Dit is een canoniek probleem. We kunnen de rijen  $F$  en  $G$  beschouwen als de gesorteerde representatie van twee verzamelingen integers. In die terminologie bepaalt `coincount` de cardinaliteit van de doorsnijding van deze twee verzamelingen. (Om der wille van de eenvoud bepaalt `coincount` slechts de cardinaliteit van de doorsnijding; bepaling van de doorsnijding zelf vergt slechts een simpele toevoeging.) Omdat de bepaling van de doorsnijding van twee verzamelingen een veel voorkomend probleem is, verklaart deze korte uitweiding tevens de populariteit van sorteerroutines.

De standaardtechniek "vervang in de eindconditie constanten door variabelen" leidt, wanneer we de symmetrie niet willen verstoren, tot de introductie van twee integer variabelen  $m$  en  $n$  en bijvoorbeeld de invariant

```

PO:    0 ≤ m ≤ M ∧ 0 ≤ n ≤ N ∧
        k = (N i, j: 0 ≤ i < m ∧ 0 ≤ j < n: F(i) = G(j)) .

```

Initialisatie is geen probleem aangezien de toestand  $(k, m, n) = (0, 0, 0)$  aan  $P_0$  voldoet en  $P_0$  is bruikbaar in de zin dat --blijkens zijn constructie-- uit

$$P_0 \wedge m = M \wedge n = N$$

de eindconditie volgt.

Het punt is evenwel dat onder herhaald verhogen van  $m$  en/of  $n$  met 1 er vele paden zijn van  $(m, n) = (0, 0)$  naar  $(m, n) = (M, N)$ , waardoor de vraag rijst of wij deze vrijheid met vrucht kunnen uitbuiten door hem te beknotten, dat wil zeggen door onze invariant te versterken.

Laten wij bijvoorbeeld proberen het pad van het punt  $(m, n)$  zo te kiezen dat elke coïncidentie in de toestand  $F(m) = G(n)$  gedetecteerd wordt. Dat betekent dat in de toestand  $F(m) \neq G(n)$  bij verhoging van  $m$  met 1 het increment van  $k$

$$(\bigvee_{j: 0 \leq j < n: F(m) = G(j)) = 0$$

zij, een conclusie die bij gratie van de monotoniciteit van  $G$  door

$$F(m) > G(n - 1)$$

is gewaarborgd. Om der wille van deze waarborg versterken we  $P_0$  --om redenen van symmetrie met twee termen-- tot  $P_1$ :

$$P_1: P_0 \wedge F(m) > G(n - 1) \wedge G(n) > F(m - 1)$$

waarbij  $F(-1)$  en  $G(-1)$  als "min oneindig" en  $F(M)$  en  $G(N)$  als "plus oneindig" erbij worden gedefinieerd.

Van onze sterkere  $P_1$  plukken we het voordeel dat de berekening niet tot  $(m, n) = (M, N)$  hoeft te worden voortgezet, want

$$P_1 \wedge (m = M \vee n = N)$$

impliceert de eindconditie reeds. (Uit  $P_1 \wedge m = M$  volgt

$G(n) > F(M - 1)$  zodat  $G(j: n \leq j < N)$  vanwege de monotonie van beide rijen geen coïncidenties oplevert; voor  $P1 \wedge n = N$  dito.)

De invariantie van  $G(n) > F(m - 1)$  impliceert dat  $m := m + 1$  als guard  $G(n) > F(m)$  krijgt, en zo komen we tot de symmetrische oplossing voor coincount

```

| [ m, n: int
  ; k := 0; m := 0; n := 0
  ; do  $m \neq M \wedge n \neq N \rightarrow$ 
    if  $G(n) > F(m) \rightarrow m := m + 1$ 
    ||  $G(n) = F(m) \rightarrow k := k + 1; m := m + 1; n := n + 1$ 
    ||  $G(n) < F(m) \rightarrow n := n + 1$ 
    fi
  od
]| .

```

\* \* \*

Gezien de eenvoud van de uiteindelijke oplossing is de weg die ertoe geleid heeft misschien wel wat lang. Er is inderdaad een kortere weg. Ter inleiding kijken we eerst even naar een één-dimensionale tellerij. Bij een eindconditie

$$c = (\underline{N} \ i: 0 \leq i < M: F(i) = 7)$$

suggereert de standaard-strategie de invariant

$$Q0: \quad 0 \leq m \leq M \wedge c = (\underline{N} \ i: 0 \leq i < m: F(i) = 7) \quad .$$

Door aan beide kanten van het gelijkteken het "ontbrekende gedeelte" op te tellen, krijgen we

$$\begin{aligned}
 Q1: \quad 0 \leq m \leq M \wedge \\
 c + (\underline{N} \ i: m \leq i < M: F(i) = 7) = \\
 (\underline{N} \ i: 0 \leq i < M: F(i) = 7) \quad .
 \end{aligned}$$

Relaties  $Q_0$  en  $Q_1$  zijn kennelijk equivalent. Tot zo ver het één-dimensionale geval.

Terugkerend naar  $\text{coincount}$ , in plaats van  $P_0$  hadden we --zie  $Q_1$  -- kunnen kiezen

$$P_2: \quad 0 \leq m \leq M \wedge 0 \leq n \leq N \wedge \\ k + (\underline{N} \ i, j: m \leq i < M \wedge n \leq j < N: F(i) = G(j)) = \\ (\underline{N} \ i, j: 0 \leq i < M \wedge 0 \leq j < N: F(i) = G(j)) \quad .$$

Evenals  $P_0$  legt  $P_2$  het pad van het punt  $(m, n)$  niet vast. Keuze van invariant  $P_2$  leidt echter haast onvermijdelijk tot de gevonden  $\text{coincount}$ .

Hadden wij om te beginnen ook aan  $P_2$  gedacht, dan hadden wij die waarschijnlijk op grond van de volgende overweging boven  $P_0$  moeten verkiezen.

Als wij initialiseren (met  $k = 0$ ) is bij  $P_2$   $m = 0 \wedge n = 0$  (zonder inspectie van de rijen) verplicht; voor  $P_0$  is  $m = 0 \vee n = 0$  voldoende, een vrijheid waarvan we om de symmetrie niet te verstoren geen gebruik hebben gemaakt. We zouden ook niet weten hoe we hem zouden moeten exploiteren.

Aan het einde ligt het precies andersom: om uit  $P_0$  de eindconditie te kunnen concluderen is de nevenconditie  $m = M \wedge n = N$  vereist, terwijl bij  $P_2$  de zwakkere  $m = M \vee n = N$  toereikend is. En: hoe zwakker de vereiste nevenconditie, hoe sterker de guard van de repetitie en hoe groter de waarschijnlijkheid op eerdere beëindiging.

De moraal van dit verhaal is dat, als wij ons bij initialisatie op vrijheid betrappen, wij er waarschijnlijk goed aan doen te onderzoeken of wij niet met vrucht een andere invariant kunnen aanhouden.

Opgave. Generaliseer de gegeven oplossing voor `coincount` voor het geval dat slechts gegeven is dat de rijen `ascending` zijn, dat wil zeggen

$$\begin{aligned} &(\underline{A} \ i, j: 0 \leq i < j < M: F(i) \leq F(j)) \wedge \\ &(\underline{A} \ i, j: 0 \leq i < j < N: G(i) \leq G(j)) \end{aligned} .$$

(Einde van Opgave.)

### De minimum "afstand"

Gevraagd een oplossing voor `mindist`, gespecificeerd door

```

| [ M, N: int {M ≥ 1 ∧ N ≥ 1}
; F(i: 0 ≤ i < M), G(j: 0 ≤ j < N): array of int
  { (A i, j: 0 ≤ i < j < M: F(i) ≤ F(j)) ∧
    (A i, j: 0 ≤ i < j < N: G(i) ≤ G(j)) ∧
    D = (MIN i, j: 0 ≤ i < M ∧ 0 ≤ j < N: abs(F(i) - G(j))) }
; | [ d: int
    ; mindist
      { d = D }
  ] |
] |

```

Om te beginnen doen we er goed aan ook voor een lege zak integers een (symbolisch) minimum te definiëren; wij kiezen hiervoor "plus oneindig", dat wil zeggen een waarde ten minste het gezochte minimum over de niet-lege zak. (Voor een lege zak is het (symbolisch) maximum op analoge wijze "min oneindig".) In dit geval kunnen wij voor "plus oneindig" bijvoorbeeld

$$\max(F(M - 1) - G(0), G(N - 1) - F(0))$$

kiezen.

Nu ook voor de lege zak een minimum gedefinieerd is, staan weer twee mogelijkheden voor de invariant open. Op grond van de moraal van het vorige voorbeeld kiezen we *niet*

$$d = \{\text{MIN } i, j: 0 \leq i < m \wedge 0 \leq j < n: \text{abs}(F(i) - G(j))\}$$

maar

$$\begin{aligned} \text{PO: } & 0 \leq m \leq M \wedge 0 \leq n \leq N \wedge \\ & \min(d, \{\text{MIN } i, j: m \leq i < M \wedge n \leq j < N: \text{abs}(F(i) - G(j))\}) \\ & = D \end{aligned}$$

Invariant PO geldt in  $(d, m, n) = (\text{"plus oneindig"}, 0, 0)$  zodat initialisatie geen probleem is. Verder geldt bij gratie van onze introductie van het symbolisch minimum over de lege zak

$$\text{PO} \wedge (m = M \vee n = N) \Rightarrow d = D$$

Voor  $0 \leq m < M \wedge 0 \leq n < N$  bekijken we twee gevallen:

$F(m) \geq G(n)$ . Voor de duidelijkheid de twee argumenten van  $\min$  onder elkaar schrijvend, herleiden we met  $K(i, j)$  voor  $\text{abs}(F(i) - G(j))$ :

$$\begin{aligned} & \min(d, \\ & \quad \{\text{MIN } i, j: m \leq i < M \wedge n \leq j < N: K(i, j)\}) = \\ & \min(d, \\ & \quad \min(\{\text{MIN } i: m \leq i < M: K(i, n)\}, \\ & \quad \quad \{\text{MIN } i, j: m \leq i < M \wedge n + 1 \leq j < N: K(i, j)\}))) = \\ & \min(d, \\ & \quad \min(K(m, n), \\ & \quad \quad \{\text{MIN } i, j: m \leq i < M \wedge n + 1 \leq j < N: K(i, j)\}))) = \\ & \min(\min(d, K(m, n)), \\ & \quad \{\text{MIN } i, j: m \leq i < M \wedge n + 1 \leq j < N: K(i, j)\}) \end{aligned}$$

In bovenstaande herleiding berust het 1<sup>ste</sup> gelijktteken op de definitie van MIN, het 2<sup>de</sup> gelijktteken voorts op de definitie van

$K(i, j)$ , op de ongelijkheid  $F(m) \geq G(n)$  en de monotonie van de  $F$ -rij, en het 3<sup>de</sup> gelijktteken op de definitie van  $\min$ .

$G(n) \geq F(m)$ . Om redenen van symmetrie analoog. Onder gebruikmaking van de functies  $\min$  en  $\max$  leidt bovenstaande analyse tot de volgende oplossing voor  $\text{mindist}$

```

| [ m, n: int
  ; d:= max(F(M - 1) - G(0), G(N - 1) - F(0))
  ; m:= 0; n:= 0
  ; do m  $\neq$  M  $\wedge$  n  $\neq$  N  $\rightarrow$ 
    if F(m)  $\geq$  G(n)  $\rightarrow$ 
      d:= min(d, F(m) - G(n)); n:= n + 1
    [] G(n)  $\geq$  F(m)  $\rightarrow$ 
      d:= min(d, G(n) - F(m)); m:= m + 1
    fi
  od
]| .

```

Her codering zonder gebruikmaking van de functies  $\min$  en  $\max$  wordt aan de lezer overgelaten.

Opmerking. Bij de herleiding hebben we gebruik gemaakt van

$$\min(A, \min(B, C)) = \min(\min(A, B), C)$$

Hadden we in plaats van de functienotatie de infixoperator min gebruikt, dan luidde deze stelling

$$A \text{ min } (B \text{ min } C) = (A \text{ min } B) \text{ min } C,$$

dat wil zeggen de infixoperator min is behalve symmetrisch ook associatief, en we hadden  $A \text{ min } B \text{ min } C$  mogen schrijven. Voor max geldt hetzelfde. In onze herleiding hebben we op gronden van conventionaliteit (ditmaal nog) van de functienotatie gebruik gemaakt. (Einde van Opmerking.)

## De maximale monotone deelrij

Van de rij  $F(i: 0 \leq i < N)$  --met  $0 \leq N$ -- heet  $F(i: k \leq i < k + h)$  "een monotone deelrij ter lengte  $h$ " indien  $Q(k, h)$  geldt, waar  $Q(k, h)$  gegeven is door

$$Q(k, h): 0 \leq k \leq k + h \leq N \wedge \\ ((\underline{A} i, j: k \leq i < j < k + h: F(i) \leq F(j)) \vee \\ (\underline{A} i, j: k \leq i < j < k + h: F(i) \geq F(j))) \quad .$$

Gevraagd een programma dat voor gegeven  $F$  de maximum lengte van enige monotone deelrij bepaalt. Formeel: gevraagd een programma `maxmonlen` dat voldoet aan de specificatie

```

| [ N: int {N ≥ 0}
  ; F(i: 0 ≤ i < N): array of int
  ; | [ q: int
    ; maxmonlen
      {q = (MAX k, h: Q(k, h): h)}
  ] |
] | .

```

(Wij raden de zelfstandige lezer van dit dictaat speciaal bij dit voorbeeld met klem aan *niet* door te lezen, het dictaat terzijde te leggen en zelf te proberen een oplossing voor `maxmonlen` te ontwerpen. Dit advies wordt gegeven omdat men slechts uit eigen ervaring leren kan hoe licht men het zichzelf onnodig moeilijk maakt. Zelfs na deze waarschuwing is de kans op die ervaring namelijk nog steeds aanzienlijk.)

Door de wijze waarop `maxmonlen` gespecificeerd is schuilen er drie addertjes onder het gras, te weten twee kleine en een grote.

Het eerste kleine addertje is gelegen in het feit dat de  $F$ -rij ook leeg mag zijn. Niet alleen dat voor  $N = 0$  na afloop evident  $q = 0$  dient te gelden,  $N = 0$  is ook het enige geval dat dit na afloop gelden kan: voor positieve  $N$  is  $q$  na afloop ten minste 1. In het algemeen proberen wij altijd zo weinig mogelijk speciale gevallen te introduceren en als het geval  $N = 0$  soepel met het algemene geval kan meelopen is dat meegenomen. Blijkt bij het algemene geval het geval  $N = 0$  een blok aan het been, dan dienen we ons te herinneren dat we het ook apart kunnen behandelen.

Het tweede kleine addertje is dat in onze eindconditie  $N$  niet zo expliciet meer voorkomt:  $N$  is als globale constante in de definitie van  $Q(k, h)$  weggemoffeld.

De grote adder is evenwel de volgende: blijkens de definitie van "monotone deelrij" is een monotone deelrij òf ascending, òf descending òf beide. Bovendien weten wij niet of de maximale lengte gerealiseerd wordt door een ascending of door een descending deelrij. De ervaring leert dat de verleiding groot is te trachten de rij  $F$  op te knippen in stijgende, dalende en constante trajecten. De veelheid van manieren waarop die elkaar kunnen opvolgen leidt evenwel tot een combinatorische explosie. Deze ellende wordt volledig vermeden wanneer de identificaties van ascending en van descending deelrijen strict gescheiden worden gehouden.

Om die scheiding door te voeren beperken we ons eerst tot de ascending deelrijen, dat wil zeggen kijken we naar het programma waarvan de eindconditie luidt

$$q = (\text{MAX } k, h: AS(k, h, N): h)$$

met  $AS$  gedefinieerd door

$$AS(k, h, n): \quad 0 \leq k \leq k + h \leq n \wedge \\ (\forall i, j: k \leq i < j < k + h: F(i) \leq F(j)) \quad .$$

Het tweede addertje hebben we inmiddels ondervangen door de introductie van  $n$ ; voorts zullen we ons gemakshalve tot  $1 \leq N$  beperken.

Als invariant kiezen we

$$q = (\text{MAX } k, h: AS(k, h, n): h) \wedge 1 \leq n \leq N .$$

Omdat

$$\begin{aligned} (\text{MAX } k, h: AS(k, h, n+1): h) = \\ (\text{MAX } k, h: AS(k, h, n): h) \max \\ (\text{MAX } k, h: AS(k, h, n+1) \wedge k+h = n+1: h) \end{aligned}$$

houden we als neveninvariant --zie de minimale segmentsom-- in stand

$$v = (\text{MAX } k, h: AS(k, h, n) \wedge k+h = n: h)$$

of, na eliminatie van  $k$ ,

$$\begin{aligned} v = (\text{MAX } h: 0 \leq n-h \leq n \wedge \\ (\text{A } i, j: n-h \leq i < j < n: F(i) \leq F(j)): h) . \end{aligned}$$

Voor het deelprobleem vinden we

```

| [ n, v: int
  ; n:= 1; q:= 1; v:= 1
  ; do n ≠ N → if F(n) ≥ F(n-1) → v:= v+1
                                     ; q:= q max v
    | F(n) < F(n-1) → v:= 1
    fi; n:= n+1
  od
]| ,

```

een oplossing die inderdaad voor  $N = 0$  niet werkt.

Na het voorafgaande volgt de oplossing voor maxmonlen --met w analoog aan v -- :

```

if N = 0 → q:= 0
  [] N > 0 →
    |[ n, v, w: int
      ; n:= 1; q:= 1; v:= 1; w:= 1
      ; do n ≠ N →
        if F(n) > F(n - 1) → w:= 1; v:= v + 1
        [] F(n) = F(n - 1) → w:= w + 1; v:= v + 1
        [] F(n) < F(n - 1) → w:= w + 1; v:= 1
        fi
      ; q:= q max v max w
      ; n:= n + 1
    od
  ]|
fi .

```

Met

```

if q ≥ v ∧ q ≥ w → skip
  [] q < v v q < w →
    if v ≥ w → q:= v [] w ≥ v → q:= w fi
fi

```

is de infixoperator max desgewenst te elimineren.

## De inversietelling

Wij beschouwen twee rijen natuurlijke getallen  $X(i: 0 \leq i < N)$  en  $Y(i: 0 \leq i < N)$  die voldoen aan de volgende betrekking

$X(i: 0 \leq i < N)$  is een permutatie van de getallen van  $0$  t/m  $N - 1$   $\wedge$

$(\underline{A} \ j: 0 \leq j < N: (\underline{N} \ i: 0 \leq i < j: X(i) < X(j)) = Y(j))$  .

Opmerking. De eerste term van deze betrekking hadden we ook formeel kunnen uitdrukken door

$$(\underline{A} \ j: 0 \leq j < N: (\underline{N} \ i: 0 \leq i < N: X(i) = j) = 1) \quad .$$

(Einde van Opmerking.)

Opgave. Ga na dat uit het bovenstaande volgt

$$(\underline{A} \ j: 0 \leq j < N: Y(j) \leq j) \quad .$$

(Einde van Opgave.)

Voor zulk een tweetal rijen is `invercount` gespecificeerd door

```

| [ N: int {N ≥ 0}
  ; | [ v(i: 0 ≤ i < N): array of int
    {v = X}
    ; invercount
    {v = Y}
  ]
]|

```

De bevrijdende opmerking is dat elk natuurlijk getal gelijk is aan het aantal kleinere natuurlijke getallen, dat wil zeggen  $(\underline{N} \ i: 0 \leq i < n) = n$  voor  $n \geq 0$ . Hieruit volgt  $X(N - 1) = Y(N - 1)$ , dat wil zeggen de waarde van  $v(N - 1)$  wordt door `invercount` ongewijzigd gelaten. Dit suggereert om de elementen van  $v$  in volgorde "van achteren naar voren" aan die van  $Y$  gelijk te maken. Als invariant kiezen we  $P0 \wedge P1 \wedge P2$  met

$P0: \quad v(i: 0 \leq i < n)$  is een permutatie van de getallen van  $0$  tot  $n - 1$

$P1: \quad (\underline{A} \ j: 0 \leq j < n: (\underline{N} \ i: 0 \leq i < j: v(i) < v(j)) = Y(j))$

$P2: \quad (\underline{A} \ j: n \leq j < N: v(j) = Y(j)) \quad .$

Uit  $v = X \wedge n = N$  volgt de invariant, uit  $P2 \wedge n = 0$  volgt  $v = Y$ .

De aflaging  $n := n - 1$  verstoort  $P2$  niet op grond van de bevrijdende opmerking en laat  $P1$  triviaal onverlet;  $P0$ , evenwel, zal in het algemeen verstoord zijn: na aflaging van  $n$  is  $v(i: 0 \leq i < n)$  een permutatie van de getallen van  $0$  t/m  $v(n) - 1$  en van  $v(n) + 1$  t/m  $n$ . Door deze laatste getallen met  $1$  te verlagen herstellen we  $P0$  zonder  $P1 \wedge P2$  te verstoren. Voor invercount vinden we

```

[[ n: int; n:= N
  ; do n  $\neq$  0  $\rightarrow$ 
    [[ i: int; i:= 0; n:= n - 1
      ; do i  $\neq$  n  $\rightarrow$  if v(i) > v(n)  $\rightarrow$  v:(i)= v(i) - 1
        || v(i) < v(n)  $\rightarrow$  skip
      fi; i:= i + 1
    ] ]
  ] ]
od
]]
.
```

Voor  $X$  zijn  $N!$  verschillende waarden mogelijk. De in de opgave genoemde eigenschap laat voor  $Y$  eveneens  $N!$  verschillende waarden toe. Als die allemaal mogelijk zijn --en zulks is inderdaad het geval-- betekent dat dat de uitvoering van invercount geen informatie vernietigt en dat er dus ook een programma bestaat dat, gegeven de  $Y$ -rij, de bijbehorende  $X$ -rij reconstrueert. De transformatie van  $v$  die door de uitvoering van invercount wordt bewerkstelligd is dan inverteerbaar.

Elk deterministisch programma  $S$ , waarvan de uitvoering geen informatie vernietigt, heeft een inverse  $S^{-1}$ . Of een text voor  $S^{-1}$  direct uit die voor  $S$  afleidbaar is, hangt af van de wijze

waarop  $S$  geprogrammeerd is. De boven gegeven oplossing voor invercount is met zorg zo gekozen dat  $\text{invercount}^{-1}$  er uit afleidbaar is.

We geven hieronder in twee kolommen geannoteerde statements die elkaars inverse zijn.

$x := x + 1$	$x := x - 1$
$S_0; S_1$	$S_1^{-1}; S_0^{-1}$
$\text{[ [ } x: \text{int}$ $\quad ; x := \text{exp1}$ $\quad ; S$ $\quad \{x = \text{exp2}\}$ $\text{] ]}$	$\text{[ [ } x: \text{int}$ $\quad ; x := \text{exp2}$ $\quad ; S^{-1}$ $\quad \{x = \text{exp1}\}$ $\text{] ]}$
$\underline{\text{if}} \ B_0 \rightarrow S_0 \ \{C_0\}$ $\quad \square \ B_1 \rightarrow S_1 \ \{C_1\}$ $\underline{\text{fi}}$ $\quad \text{met } \neg(B_0 \wedge B_1) \text{ en } \neg(C_0 \wedge C_1)$ $\{\neg C\}$ $\underline{\text{do}} \ B \rightarrow S \ \{C\} \ \underline{\text{od}}$ $\{\neg B\}$	$\underline{\text{if}} \ C_1 \rightarrow S_1^{-1} \ \{B_1\}$ $\quad \square \ C_0 \rightarrow S_0^{-1} \ \{B_0\}$ $\underline{\text{fi}}$ $\{\neg B\}$ $\underline{\text{do}} \ C \rightarrow S^{-1} \ \{B\} \ \underline{\text{od}}$ $\{\neg C\}$

We zullen nu  $\text{invercount}$  van een zodanige annotatie voorzien, dat bovenstaande inversieregels van toepassing zijn:

```

[ [ n: int; n := N
  {n = N}
; do n ≠ 0 →
  [ [ i: int; i := 0; n := n - 1
    {i = 0}
  ; do i ≠ n →
    if v(i) > v(n) → v:(i) = v(i) - 1 {v(i) ≥ v(n)}
    □ v(i) < v(n) → skip {v(i) < v(n)}
  fi; i := i + 1 {i ≠ 0}
  ] ]
] ]

```

```

        od
        {i = n}
    ]I {n ≠ N}

    od
    {n = 0}
]I .

```

Voor  $\text{invercount}^{-1}$  is de functionele specificatie

```

|I N: int {N ≥ 0}
; |I v(i: 0 ≤ i < N): array of int
    {v = Y}
; invercount-1
    {v = X}
]I
]I ;

```

er wordt aan voldaan door de --niet geannoteerde-- inverse van `invercount`:

```

|I n: int; n:= 0
; do n ≠ N →
    |I i: int; i:= n
    ; do i ≠ 0 → i := i - 1
        ; if v(i) < v(n) → skip
            [] v(i) ≥ v(n) → v:(i)= v(i) + 1
        fi
    od; n:= n + 1
]I
od
]I .

```

Opgave. Annoteer bovenstaande oplossing voor  $\text{invercount}^{-1}$ , zodat zijn inverse weer `invercount` oplevert. (Einde van Opgave.)

Historische opmerking. Programma-inversie is spelenderwijs (als een soort van programmeergrap) ontwikkeld (naar aanleiding van

bovenstaande problemen die samen een keer als tentamenopgaven gefigureerd hebben). Later is programma-inversie bij serieuze programma-ontwikkeling toepasbaar gebleken, als in een programma waarin --voor vrij ingewikkelde, recurrent gedefinieerde  $F$ -- uit efficiency-overwegingen  $f = F(q)$  als invariant werd toegevoegd. De enige wijzigingen waaraan  $q$  werd onderworpen waren  $q := q + 1$  en  $q := q - 1$ ; van de bijbehorende, vrij ingewikkelde aanpassingen van de waarde van  $f$  kon de ene door inversie uit de andere verkregen worden. (Einde van Historische opmerking.)

## De getallen met slechts factoren 2, 3 en 5

Gevraagd een programma dat in opklimmende volgorde de eerste 1000 oplossingen genereert van de vergelijking in  $x$

$$(\exists n_2, n_3, n_5: n_2 \geq 0 \wedge n_3 \geq 0 \wedge n_5 \geq 0: 2^{n_2} \cdot 3^{n_3} \cdot 5^{n_5} = x) .$$

Een andere manier om de verzameling  $V$  van oplossingen van deze vergelijking te karakteriseren is

- (i) 1 behoort tot  $V$
- (ii) als  $x$  tot  $V$  behoort behoren ook  $2 \cdot x$ ,  $3 \cdot x$  en  $5 \cdot x$  tot  $V$
- (iii) alleen de waarden die op grond van (i) en (ii) tot  $V$  behoren behoren tot  $V$ .

Zij  $V_{1000}(i: 0 \leq i < 1000)$  de "increasing" rij van de kleinste 1000 elementen van  $V$ . De functionele specificatie van hamming --zo genoemd naar R.W. Hamming die deze programmeeropgave gelanceerd heeft-- luidt dan

```

| | | x(i: 0 ≤ i < 1000): array of int
      ; hamming
      {x = V1000}
| |
| | .

```

Voor de hand ligt de invariant

P0:  $1 \leq n \leq 1000 \wedge \underline{A} \ i: 0 \leq i < n: V1000(i) = x(i)$

die op grond van (i) gemakkelijk voor  $n = 1$  te initialiseren is. Voor hamming suggereert dit een oplossing van de structuur

```

| | n: int
      ; x(0) = 1; n := 1 {P0}
      ; do n ≠ 1000 → "verhoog n met 1 onder invariantie
                        van P0"
      od
| | .

```

De opgave "verhoog n met 1 onder invariantie van P0" betekent dat de waarde van  $V1000(n)$  moet worden bepaald. Op grond van (ii) en (iii) is  $V1000(n)$  van de vorm  $2 \cdot x(i_2)$  met  $0 \leq i_2 < n$ , of van de vorm  $3 \cdot x(i_3)$  met  $0 \leq i_3 < n$  of van de vorm  $5 \cdot x(i_5)$  met  $0 \leq i_5 < n$ . (Merk op dat, omdat  $2 \cdot x$ ,  $3 \cdot x$  en  $5 \cdot x$  alle drie groter dan  $x$  zijn, lidmaatschap van  $V$  op grond van (ii) altijd afhangt van een *kleinere*  $x$  die tot  $V$  behoort. Het is deze eigenschap die ons in staat stelt elementen van  $V$  in opklimmende volgorde te genereren.)

Van de getallen van de vormen  $2 \cdot x(i_2)$ ,  $3 \cdot x(i_3)$  en  $5 \cdot x(i_5)$  moeten we de kleinste hebben die  $> x(n-1)$  is. Met andere woorden van de kleinste  $> x(n-1)$  van de vorm  $2 \cdot x(i_2)$ , de kleinste  $> x(n-1)$  van de vorm  $3 \cdot x(i_3)$  en de kleinste  $> x(n-1)$  van de vorm  $5 \cdot x(i_5)$  moeten we het minimum hebben.

Voor de *kleinste* van de vorm  $2 \cdot x(i_2)$  die  $> x(n-1)$  is, moeten de waarden  $2 \cdot x(i_2)$  in *opklimmende* volgorde onderzocht worden --zie de zogenaamde linear search-- . Omdat  $2 \cdot x$  een monotoon stijgende functie van  $x$  is en  $x(i: 0 \leq i < n)$  increasing is, betekent dit dat de waarden  $2 \cdot x(i_2)$  in volgorde van opklimmende  $i_2$  onderzocht kunnen worden. Voor de andere twee factoren geldt dezelfde opmerking, en zo vinden we voor hamming :

```

| [ n: int
  ; x:(0)= 1; n:= 1
  ; do n  $\neq$  1000  $\rightarrow$ 
    | [ i2, i3, i5: int
      ; i2:= 0; i3:= 0; i5:= 0
      ; do  $2 \cdot x(i_2) \leq x(n-1) \rightarrow i_2:= i_2 + 1$  od
      ; do  $3 \cdot x(i_3) \leq x(n-1) \rightarrow i_3:= i_3 + 1$  od
      ; do  $5 \cdot x(i_5) \leq x(n-1) \rightarrow i_5:= i_5 + 1$  od
      ; x:(n)= ( $2 \cdot x(i_2)$ ) min ( $3 \cdot x(i_3)$ ) min ( $5 \cdot x(i_5)$ )
      ; n:= n + 1
    ] |
  od
] | .

```

Hiermee is de kous evenwel niet af. De *enige* functie van de assignment  $i_2:= 0$  is het bewerkstelligen van de invariant

(A  $j: 0 \leq j < i_2: 2 \cdot x(j) \leq x(n-1)$ )

van de daarop volgende linear search . Omdat de rij  $x$  stijgend is, wordt deze invariant aan het einde van het binnenblok door  $n:= n + 1$  niet verstoord. De invariant van de linear search is derhalve ook een mogelijke invariant van de buitenste repetitie, mits we declaratie en initialisatie van  $i_2$  "naar buiten brengen". Voor  $i_3$  en  $i_5$  geldt mutatis mutandis hetzelfde, en zo komen wij tot de volgende oplossing voor hamming :

```

| [ n, i2, i3, i5: int
  ; x:(0)= 1; n:= 1; i2:= 0; i3:= 0; i5:= 0
  ; do n  $\neq$  1000  $\rightarrow$ 
      do 2 * x(i2)  $\leq$  x(n - 1)  $\rightarrow$  i2:= i2 + 1 od
    ; do 3 * x(i3)  $\leq$  x(n - 1)  $\rightarrow$  i3:= i3 + 1 od
    ; do 5 * x(i5)  $\leq$  x(n - 1)  $\rightarrow$  i5:= i5 + 1 od
    ; x:(n)=
      (2*x(i2)) min (3*x(i3)) min (5*x(i5))
    ; n:= n + 1
  od
|] .

```

Opgave. Toon aan dat, in tegenstelling tot onze eerste oplossing voor hamming, in onze laatste oplossing voor hamming de repetitieve statement

```

do 2 * x(i2)  $\leq$  x(n - 1)  $\rightarrow$  i2:= i2 + 1 od

```

vervangen zou mogen worden door de alternatieve statement

```

if 2 * x(i2)  $\leq$  x(n - 1)  $\rightarrow$  i2:= i2 + 1
  || 2 * x(i2) > x(n - 1)  $\rightarrow$  skip
fi .

```

Voor de volgende twee repetitieve statements geldt mutatis mutandis hetzelfde. (Einde van Opgave.)

De overgang van de eerste oplossing naar de tweede oplossing is een standaardtransformatie, bekend onder de naam "taking a relation outside a repetition". Hij is de facto zo standaard dat de ervaren programmeur in zo'n geval de eerste versie vaak niet eens meer opschrijft maar zich meteen afvraagt met welke extra variabelen in het de repetitie omvattende blok de ene slag van de repetitie van de voorafgaande zou kunnen profiteren. (Die directere weg is degene die we bij de ontwikkeling van minsegsum bij de introductie van y hebben gevolgd.)

## Coördinatentransformatie

In het verleden --zie p. 68-- hebben wij ons beziggehouden met square root , gespecificeerd door

```

| [ N: int {N ≥ 0}
  ; | [ a: int
    ; square root
      {a2 ≤ N ∧ (a + 1)2 > N}
    ] |
  ] | .

```

We zijn toen geëindigd met de oplossing

```

| [ b: int
  ; a:= 0; b:= 1
  ; do b * b ≤ N → b:= 2 * b od
  ; do b ≠ a + 1 →
    | [ c: int; c:= (a + b)div 2
      ; if c * c ≤ N → a:= c
        | c * c > N → b:= c
      fi
    ] |
  od
] | .

```

Deze oplossing was heel efficiënt in de zin dat het aantal slagen van de repetities evenredig met  $\log N$  was --en niet evenredig met  $\sqrt{N}$  als bij een eerdere oplossing-- . Zoals de oplossing boven is gegeven vergt elke slag wel een vermenigvuldiging, en aangezien de algemene multiplicatieve operatie doorgaans (een orde van grootte) meer tijd vergt dan additieve operaties, halveren en verdubbelen, rijst de vraag of deze algemene vermenig-

vuldigingen geëlimineerd kunnen worden. Dit kan inderdaad; we zullen de eliminatie in een aantal stappen uitvoeren.

We hadden al opgemerkt dat het verschil  $b - a$  altijd een tweemacht is en dat de "div 2" daarom door  $/ 2$  vervangen mocht worden. We doen dit en houden tevens de invariant  $d = b - a$  in stand.

```

| [ b, d: int
  ; a:= 0; b:= 1; d:= 1
  ; do  $b * b \leq N \rightarrow b:= 2 * b; d:= 2 * d$  od
  ; do  $b \neq a + 1 \rightarrow$ 
    | [ c: int; c:= (a + b)/ 2
      ; if  $c * c \leq N \rightarrow a:= c; d:= d / 2$ 
      |  $c * c > N \rightarrow b:= c; d:= d / 2$ 
      fi
    ] |
  od
] | .

```

Het binnenblok is equivalent met

```

| [ c: int; c:= (a + b)/ 2; d:= d / 2
  {a + d = c  $\wedge$  c + d = b}
  ; if  $c^2 \leq N \rightarrow a:= c$ 
  |  $c^2 > N \rightarrow b:= c$ 
  fi
] |

```

(gemakshalve hebben we ons even de exponent veroorloofd).

In de laatste alternatieve statement mogen we alle 4 de  $c$ 's door  $(a + d)$  vervangen. Maar dan wordt de lokale  $c$  helemaal niet meer gebruikt, en kan hij straffeloos worden weggelaten: het binnenblok wordt aldus gereduceerd tot

```

d := d / 2
; if (a + d)2 ≤ N → a := a + d
  [] (a + d)2 > N → b := a + d
fi .

```

Vervolgens merken wij op dat vanwege de invariant  $a = 0$ , dat wil zeggen  $b = d$ , van de eerste repetitie de guard  $b^2 \leq N$  dezelfde waarde heeft als  $d^2 \leq N$  en daardoor vervangen mag worden.

Vervolgens merken we op dat vanwege de invariant  $d = b - a$  van de tweede repetitie de guard  $b \neq a + 1$  dezelfde waarde heeft als  $d \neq 1$  en daardoor vervangen mag worden.

Maar dan mag de variabele  $b$  straffeloos worden weggelaten. Het resultaat van al deze substituties en omissies is

```

|[ d: int
; a := 0; d := 1
; do d2 ≤ N → d := 2 * d od
; do d ≠ 1 → d := d / 2
      ; if (a + d)2 ≤ N → a := a + d
        [] (a + d)2 > N → skip
      fi
    od
]| .

```

Dit was maar een opstapje om ons in staat te stellen de guard  $(a + d)^2 \leq N$  te kunnen herschrijven tot  $2 \cdot a \cdot d + d^2 \leq N - a^2$ , een vorm die ons suggereert drie variabelen  $p$ ,  $q$  en  $r$  te introduceren, die voldoen aan

$$p = a \cdot d \wedge q = d^2 \wedge r = N - a^2 .$$

Deze introductie leidt tot het volgende programma - merk op dat vanwege  $a = 0$  in de eerste repetitie de verdubbeling van  $d$  niet met een verdubbeling van  $p$  gepaard hoeft te gaan.

```

| [ p, q, r, d: int
  ; p:= 0; q:= 1; r:= N; a:= 0; d:= 1
  ; do  $d^2 \leq N \rightarrow q:= 4 * q; d:= 2 * d$  od
  ; do  $d \neq 1 \rightarrow q:= q / 4; p:= p / 2; d:= d / 2$ 
      ; if  $2 * a * d + d^2 \leq N - a^2 \rightarrow r:= r - (2 * a * d + d^2)$ 
          ; p:= p + q; a:= a + d
      []  $2 * a * d + d^2 > N - a^2 \rightarrow \text{skip}$ 
    fi
  od {a = p}
]| .

```

In de eerste repetitie is de waarde van de guard  $d^2 \leq N$  gelijk aan die van  $q \leq r$ , waardoor hij vervangen mag worden.

In de tweede repetitie is de waarde van de guard  $d \neq 1$  gelijk aan die van  $q \neq 1$ , waardoor hij vervangen mag worden.

In de alternatieve constructie is de waarde van  $2 * a * d + d^2$  gelijk aan die van  $2 * p + q$ , waardoor hij mag worden vervangen, en die van  $N - a^2$  gelijk aan die van  $r$ .

Deze vervangingen maken  $d$  overbodig en de assignments aan  $a$  kunnen worden weggelaten mits we aan het einde  $a:= p$  inlassen.

Om korthedswil is de hulpvariabele  $h$  geïntroduceerd die bij gebruik voldoet aan  $h = 2 * p + q$ .

```

| [ p, q, r: int
  ; p:= 0; q:= 1; r:= N
  ; do q ≤ r → q:= 4 * q od
  ; do q ≠ 1 →
    | [ h: int
      ; q:= q / 4; h:= p + q; p:= p / 2
      ; if h ≤ r → r:= r - h; p:= p + q
        | h > r → skip
      fi
    ] |
  od; a:= p
] | .

```

Opmerking. Met bovenstaande algoritme hoeft in een microgeprogrammeerde binaire machine de worteltrekking niet veel meer tijd te kosten dan een deling. (Einde van Opmerking.)

Wij hebben hier het transformatieproces --introdunctie van nieuwe variabelen en invarianten, herschrijving van guards en daarna eliminatie van oorspronkelijke variabelen-- in alle uitvoerigheid laten zien. Op deze wijze leidt het wel tot wat veel schrijfwerk. Wie aan deze methode van programma-ontwikkeling gewend is verricht de overgang van oude op nieuwe variabelen in één stap, zonder de versie met beide op te schrijven. Het bezwaar van herschrijving blijft nochtans aan deze methode van programma-ontwikkeling kleven.

## Naar aanleiding van een gerichte graaf

We beschouwen een graaf met  $N$  verschillende punten genummerd van  $0$  t/m  $N - 1$  en  $M$  gerichte takken. Elke tak heeft één van de  $N$  punten als "beginpunt" en één van de  $N$  punten als "eindpunt". Een tak heet een "uitgaande tak" van zijn beginpunt en een "inkomende tak" van zijn eindpunt. Het eindpunt van een tak heet vanuit het beginpunt van die tak "direct bereikbaar".

De structuur van zo'n graaf kan op verschillende manieren worden vastgelegd. De meest symmetrische manier is met behulp van twee rijen,  $b(i: 0 \leq i < M)$  en  $e(i: 0 \leq i < M)$ , zodat  $b(i)$  en  $e(i)$  het nummer van beginpunt respectievelijk eindpunt zijn van de  $i$ -de tak (in een of andere willekeurige nummering). Deze representatie is wel symmetrisch, maar voor de meeste toepassingen niet zo handig: om vast te stellen welke punten direct bereikbaar zijn vanuit punt  $k$ , moet de hele  $b$ -rij afgetast worden om vast te stellen voor welke waarden van  $i$   $b(i) = k$  geldt!

We kunnen de takken ordenen: eerst de uitgaande takken van punt  $0$ , dan de uitgaande takken van punt  $1$ , enzovoorts; takken met hetzelfde beginpunt kunnen we dan weer ordenen naar ascending nummer van hun eindpunt. Met die ordening is de informatie in de  $b$ -rij ook representeerbaar door de rij  $\text{from}(j: 0 \leq j < N + 1)$  zodat

(A  $j: 0 \leq j < N:$   
     (A  $i: \text{from}(j) \leq i < \text{from}(j + 1):$   
         punt  $e(i)$  is vanuit punt  $j$  direct bereikbaar)).

De rijen  $\text{from}$  en  $e$  leggen de structuur van de graaf vast.

Opgave. Verifieer  $\text{from}(0) = 0$  en  $\text{from}(N) = M$ ; ga voorts na dat  $\text{from}(j + 1) - \text{from}(j) =$  het aantal uitgaande takken van punt  $j$ . (Einde van Opgave.)

Als een graaf  $G$  aldus door de rijen  $\text{from}$  en  $e$  beschreven wordt geldt  $\text{UIT}(G, \text{from}, e)$ . Het tweetal  $(\text{from}, e)$  geeft snel antwoord op de vraag welke punten vanuit een punt  $k$  direct bereikbaar zijn.

Deze representatie biedt geen soelaas voor de vraag vanuit welke punten een punt  $k$  direct bereikbaar is. Als dit de veelvoorkomende vraag is, kiest men de complementaire representatie. Daarin worden de takken anders geordend: eerst de inkomende takken van punt  $0$ , dan de inkomende takken van punt  $1$ , enzovoorts, en takken met hetzelfde eindpunt in volgorde van ascending nummer van hun beginpunt. De informatie vervat in de  $e$ -rij is dan ook representeerbaar door de rij  $\text{to}(j: 0 \leq j < N + 1)$  zodat

(A  $j: 0 \leq j < N:$

(A  $i: \text{to}(j) \leq i < \text{to}(j + 1):$

vanuit punt  $b(i)$  is punt  $j$  direct bereikbaar)).

De rijen  $b$  en  $\text{to}$  leggen dan eveneens de structuur van de graaf vast; als een graaf  $G$  volgens deze conventie is beschreven geldt  $\text{IN}(G, b, \text{to})$ .

De opgave is nu een programma te ontwerpen dat uit de ene representatie van een graaf  $G$  de complementaire representatie vormt.

```

| [ M, N: int {M ≥ 0 ∧ N ≥ 1}
  ; from(j: 0 ≤ j < N + 1), e(i: 0 ≤ i < M): array of int
    {UIT(G, from, e)}
  ; | [ b(i: 0 ≤ i < M), to(j: 0 ≤ j < N + 1): array of int
    ; graphcomplement
      {IN(G, b, to)}
    ] |
  ] | .

```

Wij geven eerst een oplossing voor graphcomplement alvorens deze toe te lichten.

```

| [ i, j: int
  ; j:= 0; do j ≠ N → to:(j)= 0; j:= j + 1 od
  ; i:= 0; do i ≠ M → to:(e(i))= to(e(i)) + 1; i:= i + 1 od
  ; to:(N)= M; j:= N
  ; do j ≠ 0 → j:= j - 1; to:(j)= to(j + 1) - to(j) od
  ; | [ q(j: 0 ≤ j < N): array of int
    ; j:= 0; do j ≠ N → q:(j)= to(j); j:= j + 1 od
    ; i:= 0; j:= 0
    ; do j ≠ N → {from(j) = i}
      do i < from(j + 1) →
        b:(q(e(i)))= j
        ; q:(e(i))= q(e(i)) + 1
        ; i:= i + 1
      od; j:= j + 1
    od
  ] |
] | .

```

In afwijking van onze gewoonte zullen wij voor dit programma geen invarianten formuleren: de invariant van de laatste repetitie is wat moeizaam te formuleren en helpt niet echt bij de ontwikkeling van dit programma.

Wij weten dat uiteindelijk  $to(j + 1) - to(j)$  = het aantal inkomende takken van punt  $j$ . Maar dat is gelijk aan het aantal malen dat de waarde  $j$  in de  $e$ -rij voorkomt. Met andere woorden de waarde van  $to$  kan uit de  $e$ -rij berekend worden, en de waarde van  $from$  speelt daarbij geen rol. In de eerste twee repetities worden de tellingen uitgevoerd zodat  $t(j)$  = het aantal inkomende takken van punt  $j$ . Na aanvulling met  $to(N) = M$  wordt in de derde repetitie door verschilvorming aan  $to$  zijn uiteindelijke waarde gegeven.

Ingewikkelder wordt het in het daaropvolgende binnenblok, waarin de waarde van  $from$  in de berekening moet worden betrokken en  $b$  berekend moet worden: als de takken in de (door  $from$  en  $e$ ) gegeven volgorde in rekening worden gebracht, komen hun beginpunten kris-kras in  $b$  terecht. Het lokale array  $q$  is ten behoeve van de administratie hiervan ingevoerd:  $q(h)$  is de index in array  $b$  voor het nummer van het beginpunt van de "eerstvolgende" inkomende tak van punt  $h$ ; in het programma is steeds deze  $h = e(i)$ . Array  $q$  wordt geïnitieerd met de eerste  $N$  elementen van  $to$ ; elke keer dat onder indicering met  $q(h)$  van  $b$  een nieuw element is gedefinieerd wordt die  $q(h)$  met 1 verhoogd.

In de laatste repetitie is  $i$  het nummer --in de oorspronkelijke volgorde-- van de heersende tak;  $i$  loopt op van 0 t/m  $M$ . De waarde van  $j$  --oplopend van 0 t/m  $N$ -- is steeds het nummer van het beginpunt van de heersende tak; het is dus steeds de waarde van  $j$  die in  $b$  wordt ingevuld. De plaats --index-- waar dit gebeurt hangt af van het eindpunt van de heersende tak, dat wil zeggen van  $e(i)$ : het is namelijk  $q(e(i))$ .

Rest ons nog het oplopen van  $i$  en  $j$  passend te synchroniseren. Hier speelt array  $from$  een rol. De formule onderaan bladzijde 128 vertelt precies welke opeenvolgende  $i$ -waarden bij

elke waarde van  $j$  behoren. De buitenste repetitie maakt het vaste aantal van  $N$  slagen, in elke slag gaat  $i$  met zoveel stapjes omhoog als nodig. De annotatie  $\{from(j) = i\}$  is een stukje van de invariant van de buitenste repetitie.

Opmerking. De afspraak dat aanvankelijk takken met hetzelfde beginpunt naar eindpunt waren geordend, was overbodig. Wij hebben hem gemaakt om der wille van de symmetrie: dankzij array  $q$  zijn na afloop takken met hetzelfde eindpunt naar beginpunt geordend. (Einde van Opmerking.)

Merk op dat vermeerdering en vermindering met 1 zo goed als de enige arithmetische operaties in dit programma zijn. (Machine-ontwerpers doen er goed aan het belang van de subscriptie in vergelijking met dat van de arithmetiek niet te onderschatten.)

## Het kortste pad

We beschouwen een gerichte graaf met  $N$  punten, genummerd van 0 t/m  $N - 1$  en  $M$  takken, waarbij elke tak een positieve lengte heeft. De graaf is gegeven --zie vorige voorbeeld-- door array  $from(j: 0 \leq j < N + 1)$ , array  $e(i: 0 \leq i < M)$  en array  $d(i: 0 \leq i < M)$  zodat

```
(A j: 0 ≤ j < N:
  (A i: from(j) ≤ i < from(j + 1):
    vanuit punt j is punt e(i) direct bereik-
    baar via een tak ter lengte d(i))) .
```

Een rij takken zodat van elk tweetal opeenvolgende het eindpunt van de eerste het beginpunt van de tweede is heet een "pad"

van het beginpunt van de eerste tak van de rij naar het eindpunt van de laatste tak van de rij. Een enkele tak is ook een pad; lege paden zijn ook toegestaan: begin- en eindpunt van het pad vallen dan samen.

De lengte van een pad is gedefinieerd als de som van de lengtes van de constituerende takken. Gevraagd een programma dat een kortste pad van punt A naar punt B bepaalt.

\*   \*   \*

De lengte van een kortste pad van A naar X noemen wij kortheidshalve de "afstand van X", en wij zullen ons voorshands beperken tot de bepaling van de afstand van B. De extra administratie zodat na afloop tevens een pad bekend is, dat deze afstand als lengte heeft, komt later wel.

Als een kortste pad van A naar B via C loopt, begint het met een kortste pad van A naar C. Dit suggereert om voor de punten van de graaf hun afstanden te bepalen in de volgorde van ascending afstand. Noem de punten, waarvoor de afstand is bepaald, de zwarte punten. Zodra B zwart is, hebben we het antwoord gevonden; blijkt voordien de verzameling zwarte punten onuitbreidbaar, dan is B niet vanuit A bereikbaar.

Met wat voor punt kan de verzameling zwarte uitgebreid worden? Met een nog niet-zwart punt, maar dan wel met een niet-zwart punt dat direct vanuit een zwart punt bereikbaar is: van de niet-zwarte punten zoeken we immers eentje met de minimum afstand. De niet-zwarte punten die direct vanuit een zwart punt bereikbaar zijn, kleuren we grijs; de resterende punten zijn wit.

Als er geen grijze punten zijn is de verzameling zwarte punten niet uitbreidbaar. Als er wel grijze punten zijn, welke kiezen we

dan om zwart te maken? Kortste paden van A naar een niet-zwart punt bevatten precies 1 tak van zwart naar grijs, en alle voorgaande takken zijn van zwart naar zwart. Een pad vanuit A, beginnend met 0 of meer takken van zwart naar zwart en afgesloten met 1 tak van zwart naar grijs, noemen we een "speciaal pad" van zijn eindpunt. Een grijs punt met het kortste speciale pad mag zwart worden gemaakt en de lengte van dat kortste speciale pad is dan de afstand van dat punt.

Het is dus voldoende om per grijs punt de lengte van zijn kortste speciale pad bij te houden. We beschouwen het geval dat punt C van grijs zwart wordt.

- (i) Een uitgaande tak van C leidt naar een wit punt. Dit punt wordt grijs en zijn minimale speciale-padlengte wordt de afstand van C vermeerderd met de lengte van de tak in kwestie.
- (ii) Een uitgaande tak van C leidt naar een grijs punt. De minimale speciale-padlengte van dat grijze punt wordt het minimum van wat hij was en de afstand van C vermeerderd met de lengte van de tak in kwestie.
- (iii) Een uitgaande tak van C leidt naar een zwart punt. Deze tak is nooit deel van een kortste pad vanuit A en kan verder genegeerd worden.

Langzamerhand wordt het tijd ons af te vragen welke informatie we opslaan. Enerzijds willen we rap het antwoord kunnen berekenen op veel voorkomende vragen zoals "wat is de kleur van punt X?"; anderzijds zij het bij verandering bijwerken van deze informatie niet te bewerkelijk.

Voor de vastlegging van de kleur stellen we voor een array  $clr(i: 0 \leq i < N)$  en de conventie

$\text{clr}(i) = 0 \quad \equiv \quad \text{punt } i \text{ is wit} \quad ,$   
 $\text{clr}(i) = 1 \quad \equiv \quad \text{punt } i \text{ is grijs} \quad ,$   
 $\text{clr}(i) = 2 \quad \equiv \quad \text{punt } i \text{ is zwart} \quad .$

Hiermee is, gegeven het puntnummer, noch raadpleging noch wijziging van de kleur van het punt bewerkelijk.

Voor zwarte punten moeten we de afstand vastleggen; voor grijze punten moeten we de minimale speciale-padlengte bijhouden. Aangezien bij de overgang van grijs naar zwart de laatste de eerste wordt, representeren we deze in het zelfde array

$$\text{dist}(i: 0 \leq i < N)$$

$\text{dist}(i) = \text{ongedefinieerd als } \text{clr}(i) = 0$   
= de minimale speciale-padlengte van punt  $i$  als  
 $\text{clr}(i) = 1$   
= de afstand van punt  $i$  als  $\text{clr}(i) = 2$  .

Voor de afstandsbepaling is onze laatste plicht de bepaling van een grijs punt met minimale speciale-padlengte. Wie de grijze punten zijn ligt besloten in  $\text{clr}$ ; de extractie hiervan uit  $\text{clr}$  vergt evenwel dat het hele array  $\text{clr}$  wordt afgetast, en aangezien het aantal grijze punten in het algemeen een orde van grootte kleiner is dan  $N$  stellen wij voor de identificatie van de grijze punten met behulp van enige extra administratie te bespoedigen; dit kan met behulp van de integer variabele  $\text{grn}$  en array

$$\text{gr}(i: 0 \leq i < N) \text{ en de conventie}$$

$\text{gr}(i: 0 \leq i < \text{grn}) = \text{de rij nummers van de grijze punten (in een of andere volgorde).}$

Array  $\text{gr}$  en teller  $\text{grn}$  worden gewijzigd als de collectie grijze punten verandert:

(i) punt  $k$  wordt van wit grijs:

$gr:(grn) = k; grn := grn + 1$  ;

(ii) punt  $gr(h)$  wordt van grijs zwart:

$grn := grn - 1; gr:(h) = gr(grn)$  .

Tenslotte overwegen we de extra informatie nodig voor de bepaling van een kortste pad van  $A$  naar  $B$  . Omdat wij trachten de verzameling zwarte punten te laten groeien totdat hij  $B$  omvat, ligt het voor de hand om voor alle zwarte punten  $X$  een kortste pad van  $A$  naar  $X$  vast te leggen, en analoog voor elk grijs punt  $X$  een kortste speciaal pad van  $A$  naar  $X$  .

Als een kortste (speciaal) pad van  $A$  naar  $X$  eindigt met de tak van  $C$  naar  $X$  , is het wat  $X$  betreft voldoende bij  $X$  te registreren dat hij door  $C$  wordt voorafgegaan, mits voor  $C$  een kortste pad bekend is.

Deze voorgangeradministratie wordt bijgehouden in het array  $pred(i: 0 \leq i < N)$  voor alle grijze en alle zwarte punten met uitzondering van  $A$  , dat geen voorganger heeft. Dit laatste hindert niet omdat het kortste pad van  $A$  naar  $A$  bekend is: het is het lege pad.

Tenslotte preciseren wij uit de formele specificatie hoe het antwoord vastgelegd moet worden.

$K =$       0 als  $B$  niet vanuit  $A$  bereikbaar is  
           =      het aantal punten op het gevonden kortste pad.  
 en voorts, als  $K \geq 1$

$L =$       lengte van het gevonden kortste pad

$PAD(i: 0 \leq i < K) =$  in volgorde, beginnend met  $A$  en eindigend  
                                  met  $B$  , de punten op het gevonden kortste pad.

Uit de formele specificatie geven we de declaraties:

```

| [ A, B, N, M: int
  ; from(j:  $0 \leq j < N + 1$ ), e, d(i:  $0 \leq i < M$ ): array of int
  ; | [ K, L: int
    ; PAD(i:  $0 \leq i < N$ ): array of int
    ; shorpath
  ] |
] | .

```

De kern van shorpath volgt op pagina 138; op pagina 139 volgt een eenvoudige coda, waarin het antwoord op de gevraagde manier met behulp van de variabelen K, L en PAD wordt afgeleverd. (Merk op dat het pad in eerste instantie achterstevoren wordt opgebouwd.)

De initialisatie voor de grote repetitie bestaat uit het wit maken van alle punten gevolgd door de grijskleuring --met inachtnaam van alle conventies-- van A. De grote repetitie eindigt als de grijze punten op zijn of B zwart is; merk op, dat beide het geval kunnen zijn.

Het binnenblok bestaat uit drie gedeelten.

(i) De keuze van het grijze punt C dat zwart zal worden; variabele h is geïntroduceerd omdat C uit  $gr(i: 0 \leq i < grn)$  moet worden verwijderd, variabele min speelt de gebruikelijke rol.

(ii) Punt C wordt zwart gemaakt met inachtnaam van de conventies.

(iii) De uitgaande takken van C worden bij toerbeurt beschouwd; X is het eindpunt van zo'n tak, len is de lengte van het kortste pad vanuit A dat met tak CX eindigt. Merk op dat X en len

(Voortzetting op pagina 139.)

```

| [ grn, i: int
; clr, dist, gr, pred(j:  $0 \leq j < N$ ): array of int
; i:= 0; do i  $\neq$  N  $\rightarrow$  clr:(i)= 0; i:= i + 1 od
; clr:(A)= 1; dist:(A)= 0; gr:(0)= A; grn:= 1
; do grn > 0  $\wedge$  clr(B)  $\neq$  2  $\rightarrow$ 
    | [ h, min, C: int
    ; h:= 0; min:= dist(gr(0)); i:= 1
    ; do i  $\neq$  grn  $\rightarrow$ 
        if dist(gr(i))  $\geq$  min  $\rightarrow$  skip
        | dist(gr(i))  $\leq$  min  $\rightarrow$ 
            min:= dist(gr(i)); h:= i
        fi; i:= i + 1
    od; C:= gr(h)
    ; clr:(C)= 2; grn:= grn - 1; gr:(h)= gr(grn)
    ; i:= from(C)
    ; do i < from(C + 1)  $\rightarrow$ 
        | [ X, len: int
        ; X:= e(i); len:= dist(C) + d(i)
        ; if clr(X) = 0  $\rightarrow$ 
            clr:(X)= 1; gr:(grn)= X; grn:= grn + 1
            ; dist:(X)= len
            ; pred:(X)= C
        | clr(X) = 1  $\rightarrow$ 
            if len  $\geq$  dist(X)  $\rightarrow$  skip
            | len  $\leq$  dist(X)  $\rightarrow$ 
                dist:(X)= len
                ; pred:(X)= C
            fi
        | clr(X) = 2  $\rightarrow$  skip
        fi; i:= i + 1
    ] ]
    od
] ]
od

```

```

; if clr(B)  $\neq$  2  $\rightarrow$  K:= 0
  | clr(B) = 2  $\rightarrow$ 
    PAD:(0)= B; K:= 1
    ; do PAD(K - 1)  $\neq$  A  $\rightarrow$ 
      PAD:(K)= pred(PAD(K - 1)); K:= K + 1
    od
    ; |[ j: int; i:= 0; j:= K - 1
      ; do i < j  $\rightarrow$  PAD:swap(i, j)
        ; i:= i + 1; j:= j - 1
      od
    ]|
    ; L:= dist(B)
  fi
]| .

```

(Voortzetting van pagina 137.)

lokale constanten zijn; zij zijn om korthedswil en voor de duidelijkheid geïntroduceerd. De alternatieve statement volgt de analyse van pagina 134 op de voet; als de guard  $\text{clr}(X) = 1$  vervangen wordt door  $\text{clr}(X) \geq 1$  kan het alternatief met de guard  $\text{clr}(X) = 2$  vervallen.

Opmerking. Variabele  $i$  wordt slechts gebruikt in disjuncte repetities. We hadden van die repetities binnenblokken kunnen maken, elk met een declaratie van de "eigen  $i$ ". Dat zou wel mooi geweest zijn, we hebben het uit luiheid en (slechte?) gewoonte niet gedaan. (Einde van Opmerking.)

## De "binary search"

Het doel is vast te stellen of een gegeven waarde voorkomt in een gesorteerde rij, precieser: we zoeken een oplossing voor search , gespecificeerd door

```

| [ N: int {N > 0}
  ; W(i: 0 ≤ i < N): array of int
    {(A i, j: 0 ≤ i < j < N: W(i) ≤ W(j))}
  ; X: int
  ; | [ present: bool
    ; search
      {present ≡ (E i: 0 ≤ i < N: W(i) = X)}
  ] |
] | .

```

Als  $X$  wel in rij  $W$  voorkomt, is het onwaarschijnlijk dat we dit vaststellen zonder dat ook gevonden wordt waar  $X$  in de rij  $W$  voorkomt, dat wil zeggen zonder dat een  $i$  bepaald wordt, zodat

$$W(i) = X \quad .$$

Deze relatie is te sterk om op af te stevenen, want  $X$  hoeft niet in rij  $W$  voor te komen; als  $X$  niet in de rij voorkomt, kan  $i$  op zijn best aangeven "waar  $X$  had moeten staan", dat wil zeggen

$$W(i) < X < W(i + 1) \quad .$$

Zolang we niet weten waar we aan toe zijn, stevenen we af op

$$W(i) \leq X < W(i + 1) \quad .$$

Deze relatie is nog een beetje te sterk, want  $X$  zou heel groot of heel klein kunnen zijn. De symmetrie reeds verstoord

hebbend vangen we deze twee extreme gevallen op verschillende wijze op. Voor heel grote  $X$  breiden we de rij  $W$  in gedachten --dat wil zeggen in de relaties, maar niet in het programma zelf-- uit met een symbolisch extra element  $W(N) = \text{"plus oneindig"}$ ; voor heel grote  $X$  zijn onze ongelijkheden dan bevredigd voor  $i = N - 1$ . Voor heel kleine  $X$  verzwakken we onze relatie tot

$$R: \quad W(i) \leq X < W(i + 1) \quad \vee \quad Q$$

met  $Q$  gegeven door

$$Q: \quad (\underline{A} \ i: 0 \leq i < N: W(i) > X) \quad .$$

Inmiddels is  $R$  zo zwak dat hij als vergelijking in  $i$  altijd een oplossing heeft, maar nog sterk genoeg om de eindconditie eenvoudig te bewerkstelligen: als  $R$  eenmaal geldt, kent

$$\text{present} := W(i) = X$$

omdat de rij  $W$  ascending is aan  $\text{present}$  de gewenste waarde toe.

Aan  $R$  ontlelen we de invariant

$$P: \quad 0 \leq i < j \leq N \wedge (W(i) \leq X < W(j)) \quad \vee \quad Q) \quad .$$

Initialisatie is geen probleem omdat voor  $(i, j) = (0, N)$  relatie  $P$  geldt; blijkens zijn constructie volgt  $R$  uit  $P \wedge j = i + 1$ .

Onze analyse tot zover samenvattend komen wij tot een oplossing voor  $\text{search}$  van de volgende structuur:

```

[[ i, j: int
  ; i:= 0; j:= N {P}
  ; do j ≠ i + 1 →
    "verminder j - i onder invariantie van P"
    od {R}
  ; present:= W(i) = X
]] .
```

We gaan nu kijken hoe we met beginconditie  $P \wedge j \neq i + 1$  het verschil  $j - i$  onder invariantie van  $P$  kunnen verkleinen. Onder die beginconditie bestaat er een integer  $h$  zodat  $i < h < j$ . Met zo'n  $h$  wordt het verschil  $j - i$  zowel door  $i := h$  als door  $j := h$  verkleind. De keuze tussen deze twee mogelijkheden wordt vervolgens bepaald door de eis dat  $P$  invariant blijve. Uitwerking van  $P_h^i$ , respectievelijk  $P_h^j$ , leidt tot de guards in

if  $W(h) \leq X \rightarrow i := h \quad \parallel \quad X < W(h) \rightarrow j := h$  fi .

Rest ons slechts een keuze voor  $h$ . Met  $h := i + 1$  of  $h := j - 1$  wordt weliswaar  $i < h < j$  bewerkstelligd, maar lopen we de kans dat een groot verschil  $j - i$  met slechts 1 vermindert wordt: dit zou aanleiding geven tot een search die in het slechtste geval  $N$  slagen vergt. Veel beter is de keuze  $h := (i + j) \text{div } 2$ : hierdoor wordt het verschil  $j - i$  in elke slag praktisch gehalveerd, hetwelk een aantal slagen evenredig aan  $\log N$  garandeert.

Zo komen wij tot onze uiteindelijke oplossing:

```

[[ i, j: int
  ; i := 0; j := N {P}
  ; do  $j \neq i + 1 \rightarrow$ 
    [[ h: int
      ;  $h := (i + j) \text{div } 2 \{i < h < j\}$ 
      ; if  $W(h) \leq X \rightarrow i := h \{P\}$ 
         $\parallel X < W(h) \rightarrow j := h \{P\}$ 
      fi {P}
    ] ] {P}
  od {R}
  ; present := W(i) = X
]] .
```

Opmerking 0. Merk op dat de gesorteerdheid van de rij  $W$  pas bij de assignment aan  $present$  in de beschouwing betrokken hoeft te worden. Menig behandeling in de literatuur wordt ontsierd door de keuze van een invariant voor de repetitie, waarin de gesorteerdheid van  $W$  reeds uitgebreid is betrokken; dit geeft aanleiding tot een nodeloos bewerkelijk correctheidsbewijs. (Einde van Opmerking 0.)

Opmerking 1. Merk op dat het beëindigingsbewijs van de repetitie volledig onafhankelijk is van  $W$  en  $X$ : had de alternatieve statement

$$\text{if true} \rightarrow i := h \quad \parallel \quad \text{true} \rightarrow j := h \quad \text{fi}$$

geluid, de repetitie zou nog eindigen. (Einde van Opmerking 1.)

Opmerking 2. Merk op dat de correctheid van onze oplossing niet afhangt van hoe bij div de afronding zo nodig plaatsvindt. We zijn dan ook volledig vrij de assignment aan  $h$  te vervangen door

$$h := i + (j - i) \text{div } 2 \quad ,$$

een versie die met het oog op capaciteitsoverschrijding van de integer arithmetiek eventueel de voorrang verdient. Versies waarin de beëindiging wel kritisch afhangt van afrondingsconventies zijn in de literatuur helaas niet ongebruikelijk. (Einde van Opmerking 2.)

Opmerking 3. In de literatuur vindt men vaak versies waarin de repetitie "voortijdig" beëindigd kan worden. De facto komt dit neer op een versterking van zijn guard tot

$$j \neq i + 1 \wedge W(i) \neq X \quad .$$

Deze zogenaamde "versnelling" zet echter geen zoden aan de dijk. Als  $X$  wel in de rij  $W$  voorkomt, is de verwachtingswaarde van de winst 1 slag, anders nihil. Elke slag is echter ingewikkelder. Het is niet eenvoudig om op een logaritmische algoritme nog iets te verdienen! (Einde van Opmerking 3.)

Opmerking 4. Variabele  $j$ , die alleen maar gebruikt wordt om  $R$ , waar hij niet in voorkomt, te bewerkstelligen had in een extra binnenblok gedeclareerd kunnen worden:

```

| [ i: int
  ; | [ j: int; i:= 0; j:= N
    ; do  $j \neq i + 1 \rightarrow \dots\dots\dots$  od
  ] | {R}
  ; present:= W(i) = X
]| ..... (Einde van Opmerking 4.)

```

Opmerking 5. Wij observeren ten eerste dat in de statements van search  $W$  slechts voorkomt in de vormen  $W(h)$  en  $W(i)$ ; wij observeren ten tweede de ongelijkheden  $0 < h < N$  en  $0 \leq i < N$ . Samenvattend concluderen wij dat  $W(N)$  *niet* in de berekening voorkomt en dat de karakterisering "symbolisch" dus alleszins gerechtvaardigd is. (Einde van Opmerking 5.)

De binary search is in deze collectie voorbeelden opgenomen omdat hij een belangrijke, welhaast canonieke algorithm is. Hij behore tot in al zijn details tot de parate kennis van elke ontwikkelde informaticus.

### De langste "upsequence"

Voor  $N \geq 1$  beschouwen wij de integer rij  $A(i: 0 \leq i < N)$ . De volgorde van oplopende subscript zullen wij aanduiden als "de volgorde van links naar rechts".

Voor elke  $s$  die voldoet aan  $0 \leq s \leq N$  kunnen wij een zogenaamde "deelrij ter lengte  $s$ " vormen, door een willekeurig

$(N - s)$ -tal uit de rij  $A$  te verwijderen en de resterende  $s$  elementen *in hun oorspronkelijke volgorde* te handhaven. Een deelrij ter lengte  $s$  die ascending is heet een "upsequence" ter lengte  $s$ . (Merk op dat alle deelrijen ter lengte 1 en ook de lege deelrij upsequences zijn.)

Gevraagd wordt een algoritme ter bepaling van de maximale lengte van enige upsequence vervat in een rij  $A(i: 0 \leq i < N)$ . (Merk op dat deze maximum lengte door meer dan 1 upsequence gerealiseerd kan worden: bijvoorbeeld de reeks  $(3, 1, 1, 2, 5, 3)$  levert een maximale lengte 4, gerealiseerd door zowel  $(1, 1, 2, 5)$  als  $(1, 1, 2, 3)$ .)

Onze eindconditie zij  $R$ , gegeven door

$R: \quad k = \text{de maximale lengte van enige upsequence vervat in}$   
 $A(i: 0 \leq i < N)$ .

Het is niet zo moeilijk zich ervan te overtuigen dat elk element van de  $A$ -rij in de berekening moet worden betrokken, en wij maken de (bescheiden) veronderstelling dat dit in de volgorde van links naar rechts zal gebeuren. Met andere woorden wij stellen voor de introductie van een tweede variabele  $n$  en de invariant  $P1$ , gegeven door

$P1: \quad 1 \leq n \leq N \wedge$   
 $k = \text{de maximale lengte van enige upsequence vervat in}$   
 $A(i: 0 \leq i < n),$

en een programma van de structuur

```

| [ n: int
  ; "bewerkstellig  $P1$  voor  $n = 1$ "
  ; do  $n \neq N \rightarrow$ 
    "verhoog  $n$  met 1 onder invariantie van  $P1$ "
  od
]| .

```

Initialisatie is geen probleem aangezien de toestand  $(n, k) = (1, 1)$  aan  $P1$  voldoet. De verhoging  $n := n + 1$  kan evenwel een aanpassing van de waarde van  $k$  --de enige andere variabele die in  $P1$  voorkomt!-- vergen. Omdat het begin van een upsequence weer een upsequence is, heeft de aanpassing van  $k$ , indien nodig, de vorm  $k := k + 1$  en "verhoog  $n$  met 1 onder invariantie van  $P1$ " krijgt zodoende de vorm

```

{P1  $\wedge$   $1 \leq n < N$ }
  if ...  $\rightarrow k := k + 1$   $\parallel$  ...  $\rightarrow$  skip fi
;  $n := n + 1$  {P1} .

```

Resteert slechts de opgave de guards te bepalen, dat wil zeggen de voorwaarden te bepalen waaronder  $k$  met 1 verhoogd, respectievelijk gelijk gelaten moet worden opdat na de daaropvolgende verhoging  $n := n + 1$   $P1$  gegarandeerd weer geldt.

Omdat  $A(n)$  het volgende element is dat in de beschouwing betrokken wordt kunnen we de guards als volgt invullen

```

{P1  $\wedge$   $1 \leq n < N$ }
  if  $m \leq A(n) \rightarrow k := k + 1$   $\parallel$   $A(n) < m \rightarrow$  skip fi
;  $n := n + 1$ 

```

mits de waarde van  $m$  gedefinieerd is door

D:  $m$  = het minimale meest-rechtse element van enige upsequence ter lengte  $k$  vervat in  $A(i: 0 \leq i < n)$  .

Met andere woorden om  $P1$  invariant te houden hebben we behalve  $k$  nog een variabele  $m$  nodig, een tweede derivaat van  $A(i: 0 \leq i < n)$  .

Introductie van  $m$  en keuze van  $P1 \wedge D$  als invariant leidt tot

```

| [ n, m: int
  ; k:= 1; n:= 1; m:= A(0)
  ; do n ≠ N →
      if m ≤ A(n) → k:= k + 1; m:= A(n)
      [] A(n) < m → ...
      fi; n:= n + 1
    od
  ] | .

```

Merk op dat nu  $n$  met 1 verhoogd moet worden onder invariantie van  $P1 \wedge D$ . In het eerste alternatief is de invariantie van  $D$  geen probleem: alle upsequences van de nieuwe maximale lengte hebben  $A(n)$  als meest-rechtse element en dat is dus de nieuwe waarde van  $m$ .

Maar hoe staan de zaken in het tweede alternatief, als  $A(n) < m$ ? Het nieuwe element  $A(n)$  kan dan niet gebruikt worden om een langere upsequence te vormen, maar misschien wel om de waarde van  $m$  te verlagen. Op de plaats van de laatste "..." kome voor de aanpassing van  $m$

```

{A(n) < m}
if m' ≤ A(n) → m:= A(n)
[] A(n) < m' → skip
fi

```

mits de waarde van  $m'$  is gedefinieerd door

```

D':   m' = if k = 1 → "min oneindig"
      [] k > 1 → het minimale meest-rechtse element van
                  enige upsequence ter lengte k - 1
                  vervat in A(i: 0 ≤ i < n)
      fi .

```

Met andere woorden, om  $D$  invariant te houden hebben we  $m'$

als volgend derivaat van  $A(i: 0 \leq i < n)$  nodig. Na de introductie van  $m'$  als nieuwe variabele en  $P1 \wedge D \wedge D'$  als nieuwe invariant, zullen we voor de invariantie van  $D'$  een  $m''$  nodig hebben, enzovoorts. De conclusie is dat we niet een scalar  $m$  maar een array  $m$  nodig hebben, gegeven door  $P2$  --die  $D \wedge D' \wedge D'' \wedge \dots$  vervangt--

$P2: \quad (\underline{A} \ j: 1 \leq j \leq k:$   
 $\quad \quad m(j) = \text{het minimale meest-rechtse element van}$   
 $\quad \quad \text{enige upsequence ter lengte } j \text{ vervat}$   
 $\quad \quad \text{in } A(i: 0 \leq i < n) ) .$

Met invariant  $P1 \wedge P2$  is de macroscopische structuur van ons programma

```

| [ n: int
  ; m(i: 1 ≤ i < N + 1): array of int
  ; k:= 1; n:= 1; m:{1}= A(0) {P1 ∧ P2}
  ; do n ≠ N → "verhoog n met 1 onder invariantie van
    P1 ∧ P2"
    od
  ] | .

```

Uit  $P2$  volgt dat  $m(1)$  het minimum is van  $A(i: 0 \leq i < n)$  en voorts dat  $m(j: 1 \leq j \leq k)$  ascending is. Uit deze observaties volgt --ga dit na!-- hoe  $m$  moet worden aangepast als  $k$  constant blijft: "verhoog  $n$  met 1 onder invariantie van  $P1 \wedge P2$ " laat zich uitwerken tot

```

if m(k) ≤ A(n) → k:= k + 1; m:{k}= A(n)
  | A(n) < m(1) → m:{1}= A(n)
  | m(1) ≤ A(n) ∧ A(n) < m(k) →
    | [ j: int
      ; "bepaal j zodat m(j - 1) ≤ A(n) < m(j)"

```

```

      ; m:(j)= A(n)
    ]I
  fi; n:= n + 1 .

```

Voor "bepaal  $j$  zodat  $m(j - 1) \leq A(n) < m(j)$ " doen we ten slotte een beroep op de kern van de binary search, precieser: we handhaven de invariant

$$1 \leq i < j \leq k \wedge m(i) \leq A(n) < m(j) .$$

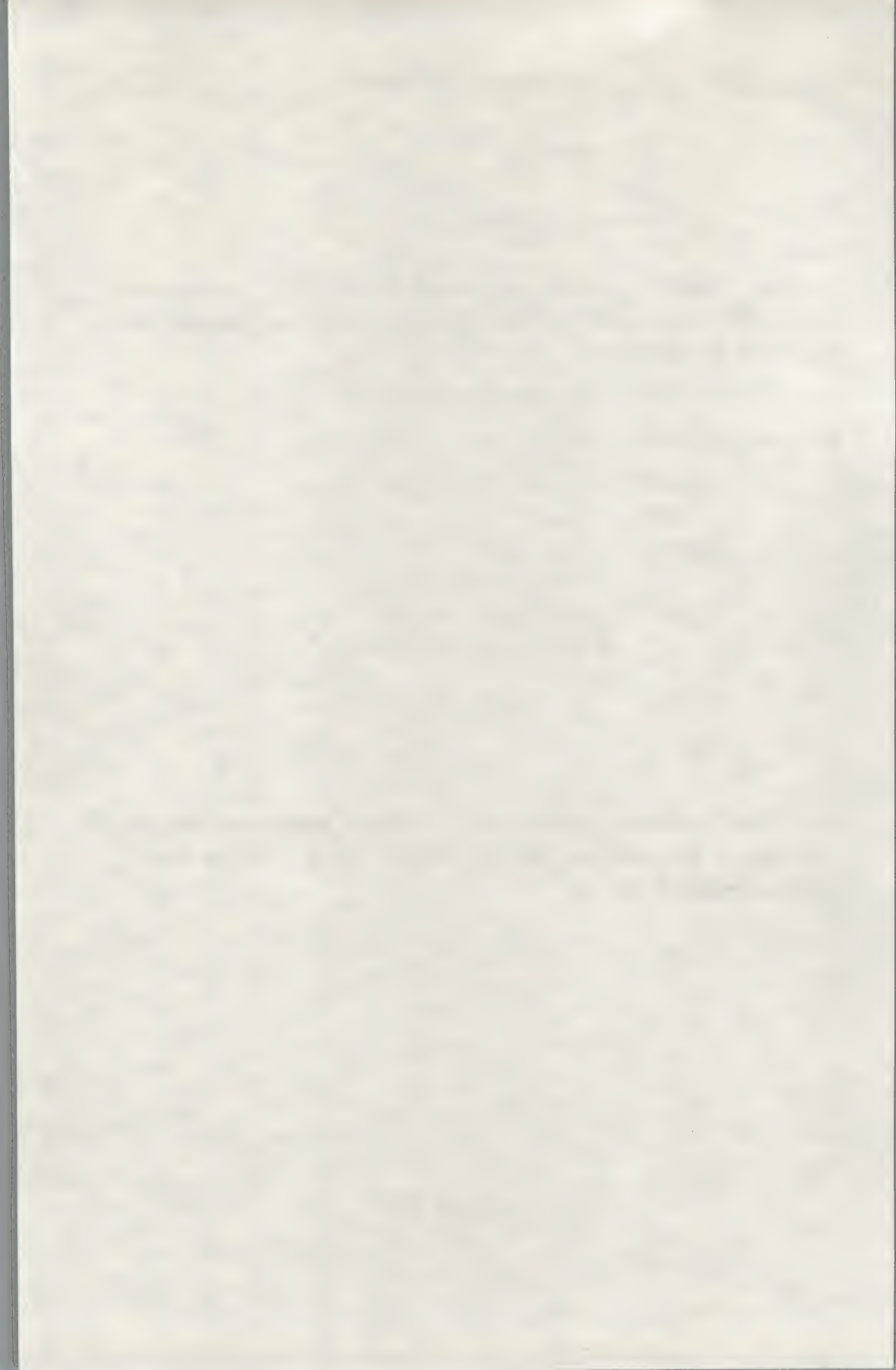
De uitwerking wordt

```

      {m(1) ≤ A(n) < m(k) ∧ k ≥ 2}
    I[ i: int; i:= 1; j:= k
      ; do j - 1 ≠ i →
        I[ h: int; h:= (i + j)div 2
          ; if m(h) ≤ A(n) → i:= h
            ∥ A(n) < m(h) → j:= h
          fi
        ]I
      ]I
    od
  ]I .

```

En hiermee is het probleem met een  $N \cdot \log N$  algoritme opgelost. Merk op dat in de extreme gevallen ( $k = 1$  of  $k = N$ ) de reken-tijd evenredig is met  $N$ .



**DEEL 1**

## 0 ALGEMENE INLEIDING

### 0.0 Predicatenrekening

Voor een formele behandeling van het begrip "predicaat" verwijzen wij naar de logica. Wij beperken ons tot een korte, informele introductie.

Daarentegen zullen wij uitvoerig ingaan op de mogelijkheid met predicaten te rekenen. Daartoe zullen wij een vrij groot aantal identiteiten behandelen en aangeven hoe met behulp hiervan een rekenspel bedreven kan worden.

Een predicaat is een expressie die wij mogen opvatten als een Boolese functie. De domeinen van predicaat en bijbehorende Boolese functie zijn dezelfde. In het merendeel van onze beschouwingen is het domein het Cartesisch product van een aantal met coördinaten benoemde (oneindige) verzamelingen. De namen van de coördinaten mogen in het predicaat voorkomen. Aan deze namen zullen we vaak refereren als "variabelen".

Veelal zijn de predicaten die wij beschouwen totaal, i.e. zijn de bijbehorende Boolese functies totale functies. Slechts indien zij partieel zijn, i.e. niet totaal, zullen wij dat expliciet vermelden.

Voorbeelden van (totale) predicaten op "het  $(x, y)$ -vlak" zijn

$$x = y \quad , \quad x^2 + y^2 \geq 1 \quad , \quad x \geq 0 \quad .$$

Een predicaat is true in een punt van het domein betekent dat de

bijbehorende Boolese functie in dat punt de waarde `true` heeft. Een predicaat is `false` in een punt van het domein als het er niet `true` is.

Vanzelfsprekend mogen wij een predicaat ook opvatten als een Boolese functie van meer variabelen dan er in de expressie voorkomen. Als gevolg daarvan zijn op elk (niet-leeg) domein twee (constante) predicaten gedefinieerd, namelijk

`true` , dat in elk punt `true` is, en

`false` , dat in elk punt `false` is .

\* \* \*

Wij beschouwen thans een vast domein en een (grote) collectie predicaten op dit domein. Met behulp van de symbolen

$\equiv$  , de equivalentie,

$\wedge$  , de conjunctie,

$\vee$  , de disjunctie,

$\neg$  , de negatie,

$\Rightarrow$  , de implicatie,

construeren wij nieuwe predicaten uit reeds bestaande.

De symbolen  $\equiv$  ,  $\wedge$  ,  $\vee$  en  $\Rightarrow$  worden gebruikt als binaire infix-operatoren en het symbool  $\neg$  als unaire prefix-operator. Van de binaire operatoren zijn  $\equiv$  ,  $\wedge$  en  $\vee$  zowel symmetrisch als associatief en is  $\Rightarrow$  noch symmetrisch, noch associatief.

Het verband tussen een aldus samengesteld predicaat en de samenstellende predicaten is, dat voor alle predicaten  $P$  ,  $Q$  en  $R$  geldt

- $P \equiv Q \equiv R$  is `true` in elk punt van het domein waar een even aantal der operanden (hier:  $P$  ,  $Q$  en  $R$ ) `false` is, en `false` elders;
- $P \wedge Q \wedge R$  is `true` in elk punt van het domein waar elk der operanden `true` is, en `false` elders;

- $P \vee Q \vee R$  is false in elk punt van het domein waar elk der operanden false is, en true elders;
- $\neg P$  is true in elk punt van het domein waar  $P$  false is, en false elders;
- $P \Rightarrow Q$  is false in elk punt van het domein waar  $P$  true is en  $Q$  false, en true elders.

Zowel ter vermindering van ambigüiteiten als ter reductie van het aantal daartoe benodigde haakjes in formules, wordt aan de operatoren een kleeftkracht toegekend:  $\neg$  heeft de grootste kleeftkracht, daarna volgen  $\wedge$  en  $\vee$ , en tenslotte  $\equiv$  en  $\Rightarrow$ . Dit betekent dat  $\neg P \vee Q \equiv \neg P$  gelezen dient te worden als  $((\neg P) \vee Q) \equiv \neg P$ . Dit betekent ook dat  $P \wedge Q \vee R$  niet opgeschreven mag worden, omdat niet duidelijk is welke van de formules  $(P \wedge Q) \vee R$  of  $P \wedge (Q \vee R)$  bedoeld wordt.

\*       \*       \*

In de collectie van alle predicaten die wij nu mogen opschrijven zijn wij in het bijzonder geïnteresseerd in de subklasse der zogeheten "geldige predicaten". Een predicaat is geldig betekent dat het in elk punt van het domein true is. Voordat wij overgaan tot het vermelden van een hele reeks geldige predicaten, geven wij eerst de voor al het rekenwerk uiterst vitale

#### Vervangingsregel

Geldt  $P \equiv Q$  en wordt een aantal voorkomens van  $P$  in de expressie  $R$  vervangen door  $Q$ , dan wordt hiermee de (on)geldigheid van  $R$  niet aangetast.

(Einde Vervangingsregel.)

Een aantal elementaire, geldige predicaten is

- (0)      $\text{true}$
- (1)      $P \equiv P \equiv \text{true}$
- (2)      $P \wedge P \equiv P$
- (3)      $P \vee P \equiv P$
- (4)      $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
- (5)      $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
- (6)      $P \wedge \text{true} \equiv P$
- (7)      $P \vee \text{true} \equiv \text{true}$
- (8)      $P \wedge \text{false} \equiv \text{false}$
- (9)      $P \vee \text{false} \equiv P$ .

Ontleden wij (1) als  $P \equiv (P \equiv \text{true})$  dan mogen wij volgens de Vervangingsregel in elk predicaat waarin " $P \equiv \text{true}$ " voorkomt deze combinatie vervangen door " $P$ ", en ook omgekeerd, zonder daarbij de (on)geldigheid van  $R$  aan te tasten. Deze eigenschap verwoorden wij doorgaans door te zeggen dat  $\text{true}$  het eenheids-element van de operator  $\equiv$  is.

Hiermee leiden we uit de geldigheid van (1) de geldigheid van

$$(10) \quad P \equiv P$$

af.

Verder volgt uit de geldigheid van (7) de geldigheid van

$$(11) \quad P \vee \text{true}.$$

Uit (6) blijkt dat  $\text{true}$  het eenheidselement van de conjunctie is, en uit (9) dat  $\text{false}$  het eenheidselement van de disjunctie is.

De geldigheid van (2) en (3) verwoorden wij meestal als de idempotentie van de conjunctie respectievelijk de disjunctie.

Aan (4) refereren wij meestal als de eigenschap dat de conjunctie

over de disjunctie distribueert, en aan (5) als de eigenschap dat de disjunctie over de conjunctie distribueert.

\*       \*       \*

Wij gaan nu over tot de karakterisering van wat wij een "berekening" zullen noemen.

Veronderstel eens dat wij op grond van Overweging 0 besluiten tot de geldigheid van  $P \equiv Q$ , en op grond van Overweging 1 tot de geldigheid van  $Q \equiv R$ ; dan volgt met de Vervangingsregel (zelfs op twee manieren) de geldigheid van  $P \equiv R$ .

Onder andere om korthedswille, geven wij deze redenering weer als de volgende, zogenaamde, berekening:

```

P
= {Overweging 0}
Q
= {Overweging 1}
R .

```

Hieruit volgt dan de geldigheid van  $P \equiv R$ , zonder de Vervangingsregel nog expliciet te noemen.

Als voorbeeld leiden we (de geldigheid van) de tweede van de twee absorptieregels

$$(12) \quad P \wedge (P \vee Q) \equiv P$$

$$(13) \quad P \vee (P \wedge Q) \equiv P$$

af:

```

P v (P ^ Q)
= {(6)}
(P ^ true) v (P ^ Q)
= {(4)}
P ^ (true v Q)
= {(7)}
P ^ true

```

= { (6) }

P .

Merk op dat wij in elke stap (Overweging) in bovenstaande berekening gebruik hebben gemaakt van de Vervangingsregel en van de geldigheid van (10). Aangezien dit steeds zo zal zijn zullen we dit nimmer vermelden.

Opgave 0 Toon (12) aan door berekening.

(Einde Opgave 0.)

\* \* \*

Wij vervolgen onze reeks geldige predicaten:

$$(14) \quad \text{false} \equiv \neg \text{true}$$

$$(15) \quad P \wedge \neg P \equiv \text{false}$$

$$(16) \quad P \vee \neg P \equiv \text{true}$$

$$(17) \quad \neg \neg P \equiv P$$

$$(18) \quad \neg(P \wedge Q) \equiv \neg P \vee \neg Q$$

$$(19) \quad \neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

$$(20) \quad \neg(P \equiv Q) \equiv \neg P \equiv Q \quad .$$

Regel (16) staat bekend als het "tertium non datur", de "wet" van de uitgesloten derde; regel (17) is de "wet" van de dubbele ontkenning; de regels (18) en (19) zijn de "wetten" van de Morgan. Regel (20) zegt dat de haakjes in die formule er niet toe doen.

Van belang zijn de twee complementregels

$$(21) \quad P \wedge Q \equiv (P \vee \neg Q) \wedge Q$$

$$(22) \quad P \vee Q \equiv (P \wedge \neg Q) \vee Q \quad .$$

Ze volgen uit de eerder genoemde regels op allerlei manieren.

Bijvoorbeeld volgt (22) uit

$$\begin{aligned}
 & P \vee Q \\
 = & \{(6)\} \\
 & (P \vee Q) \wedge \text{true} \\
 = & \{(16)\} \\
 & (P \vee Q) \wedge (\neg Q \vee Q) \\
 = & \{(5)\} \\
 & (P \wedge \neg Q) \vee Q .
 \end{aligned}$$

\*       \*       \*

Wij vervolgen onze reeks geldige predicaaten met twee regels die een verband leggen tussen  $\equiv$ ,  $\wedge$  en  $\vee$ :

$$(23) \quad (P \equiv Q) \vee R \equiv P \vee R \equiv Q \vee R$$

$$(24) \quad P \wedge Q \equiv P \equiv Q \equiv P \vee Q .$$

Formule (23) zegt dat de disjunctie distribueert over de equivalentie. Zulks geldt niet voor de conjunctie. Voor de conjunctie geldt wel

$$(25) \quad (P \equiv Q) \vee \neg R \equiv P \wedge R \equiv Q \wedge R ,$$

immers

$$\begin{aligned}
 & (P \equiv Q) \vee \neg R \\
 = & \{(17)\} \\
 & \neg \neg (P \equiv Q) \vee \neg R \\
 = & \{\text{twee keren (20)}\} \\
 & (\neg P \equiv \neg Q) \vee \neg R \\
 = & \{(23)\} \\
 & \neg P \vee \neg R \equiv \neg Q \vee \neg R \\
 = & \{\text{twee keren (18)}\} \\
 & \neg (P \wedge R) \equiv \neg (Q \wedge R) \\
 = & \{\text{twee keren (20)}\} \\
 & P \wedge R \equiv Q \wedge R .
 \end{aligned}$$

Uit (23) volgt

$$(26) \quad \neg P \vee Q \equiv P \vee Q \equiv Q$$

en uit (25)

$$(27) \quad \neg P \vee Q \equiv P \wedge Q \equiv P \quad .$$

Opgave 1 Toon (26) en (27) aan.

(Einde Opgave 1.)

Uit (26) en (27) volgt (24).

We bewijzen vervolgens de geldigheid van

$$(28) \quad P \equiv Q \equiv (\neg P \vee Q) \wedge (\neg Q \vee P) \quad :$$

$$P \equiv Q$$

$$= \{(24)\}$$

$$P \wedge Q \equiv P \vee Q$$

$$= \{(1)\} \quad .$$

$$P \wedge Q \equiv P \equiv P \equiv P \vee Q$$

$$= \{(26)\}$$

$$P \wedge Q \equiv P \equiv \neg Q \vee P$$

$$= \{(27)\}$$

$$\neg P \vee Q \equiv \neg Q \vee P$$

$$= \{(24), \text{ met } \neg P \vee Q \text{ voor } P \text{ en } \neg Q \vee P \text{ voor } Q\}$$

$$(\neg P \vee Q) \wedge (\neg Q \vee P) \equiv \neg P \vee Q \vee \neg Q \vee P$$

$$= \{(16), (7) \text{ en } (1)\}$$

$$(\neg P \vee Q) \wedge (\neg Q \vee P) \quad .$$

Opgave 2 Toon aan dat  $P \equiv Q \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q) \quad .$

(Einde Opgave 2.)

Opgave 3 Toon aan dat  $(\neg P \vee Q) \wedge (P \vee R) \equiv (P \wedge Q) \vee (\neg P \wedge R) \quad .$

(Einde Opgave 3.)

\* \* \*

Wij besluiten met de implicatie. Er geldt

$$(29) \quad (P \Rightarrow Q) \equiv P \wedge Q \equiv P ,$$

waaruit dan met (27) volgt

$$(30) \quad (P \Rightarrow Q) \equiv \neg P \vee Q ,$$

en hieruit met (26)

$$(31) \quad (P \Rightarrow Q) \equiv P \vee Q \equiv Q .$$

Uit (28) en (30) leiden we af

$$(32) \quad P \equiv Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P) .$$

Opgave 4 Toon door berekening de geldigheid aan van

$$(P \Rightarrow Q) \vee R \equiv (P \vee R \Rightarrow Q \vee R) , \text{ en van}$$

$$(P \Rightarrow Q) \vee \neg R \equiv (P \wedge R \Rightarrow Q \wedge R) .$$

(Einde Opgave 4.)

In hetgeen volgt zullen wij ons regelmatig beroepen op (de geldigheid van)

$$(33) \quad \text{false} \Rightarrow P$$

$$(34) \quad P \Rightarrow \text{true}$$

$$(35) \quad P \wedge Q \Rightarrow P$$

$$(36) \quad P \Rightarrow P \vee Q$$

$$(37) \quad (\text{true} \Rightarrow P) \equiv P .$$

We zeggen dat "P sterker is dan Q" of "Q zwakker is dan P" indien geldt  $P \Rightarrow Q$ .

Blijkens (33) is false sterker dan elk predicaat en wegens (34) is elk predicaat sterker dan true.

Blijkens (35) kan men een conjunctie verzwakken door het weglaten van een conjunct, en wegens (36) kan men een disjunctie versterken door het weglaten van een disjunct.

\* \* \*

Gelden  $P \Rightarrow Q$  en  $Q \Rightarrow R$ , dan geldt ook  $P \Rightarrow R$ , immers

$$\begin{aligned}
 &P \\
 = &\{P \Rightarrow Q \text{ geldt, en derhalve ook } P \equiv P \wedge Q\} \\
 &P \wedge Q \\
 = &\{Q \Rightarrow R \text{ geldt, en derhalve ook } Q \equiv Q \wedge R\} \\
 &P \wedge Q \wedge R \\
 = &\{P \Rightarrow Q \text{ geldt, en derhalve ook } P \wedge Q \equiv P\} \\
 &P \wedge R,
 \end{aligned}$$

zodat  $P \equiv P \wedge R$  geldt, en derhalve ook  $P \Rightarrow R$ .

Onder andere om korthedswille vatten we bovenstaande redenering samen in een berekening:

$$\begin{aligned}
 &P \\
 \Rightarrow &\{\text{argument waarom } P \Rightarrow Q \text{ geldt}\} \\
 &Q \\
 \Rightarrow &\{\text{argument waarom } Q \Rightarrow R \text{ geldt}\} \\
 &R,
 \end{aligned}$$

met als conclusie dat  $P \Rightarrow R$  geldt.

Ter illustratie tonen we aan dat

$$(38) \quad (P \Rightarrow Q) \Rightarrow (P \vee R \Rightarrow Q \vee R) :$$

$$\begin{aligned}
 &P \Rightarrow Q \\
 \Rightarrow &\{(38), \text{ met } P \Rightarrow Q \text{ voor } P \text{ en } R \text{ voor } Q\} \\
 &(P \Rightarrow Q) \vee R \\
 = &\{\text{Opgave 4}\} \\
 &P \vee R \Rightarrow Q \vee R.
 \end{aligned}$$

\* \* \*

Het is geenszins de bedoeling dat de lezer alle besproken formules van buiten leert, maar wel dat hij ermee leert omspringen. De oefeningen die volgen kunnen daarbij behulpzaam zijn.

Nadat, i.e. echt nadat, de lezer voldoende vertrouwd is geraakt met formules van bovenstaande soort kan hij in berekeningen referenties ernaar (tussen de accolades) afdoen met "elementair".

Opgave 5 Laat zien dat

- (i) uit de geldigheid van  $P$  en van  $P \equiv Q$  volgt de geldigheid van  $Q$
- (ii) uit de geldigheid van  $P \wedge Q$  volgt de geldigheid van  $P$
- (iii) uit de geldigheid van  $P$  en van  $P \Rightarrow Q$  volgt de geldigheid van  $Q$ .

(Einde Opgave 5.)

Opgave 6 Toon door berekening aan dat

- (i)  $P \wedge Q \equiv \text{false} \equiv \neg P \vee \neg Q$
- (ii)  $P \wedge Q \equiv \neg P \wedge \neg Q \equiv P \equiv \neg Q$
- (iii)  $P \wedge R \equiv Q \wedge R \equiv \neg P \wedge R \equiv \neg Q \wedge R$
- (iv)  $P \vee R \equiv Q \vee R \equiv \neg P \vee R \equiv \neg Q \vee R$ .

(Einde Opgave 6.)

Opgave 7 Toon door berekening aan dat

- (i)  $(\neg P \vee Q) \wedge (\neg Q \vee R) \wedge P \wedge \neg R \equiv \text{false}$
- (ii)  $(P \wedge Q) \vee (R \wedge S) \vee \neg P \vee \neg R \vee \neg(Q \vee S)$
- (iii)  $\neg P \vee Q \vee ((P \vee Q) \wedge (\neg P \vee \neg Q))$
- (iv)  $(P \wedge Q) \vee (Q \wedge R) \vee (R \wedge P) \equiv$   
 $(P \vee Q) \wedge (Q \vee R) \wedge (R \vee P).$

(Einde Opgave 7.)

Opgave 8 Toon door berekening aan dat

$$(i) \quad (P \equiv Q) \Rightarrow (P \Rightarrow Q)$$

$$(ii) \quad (P \Rightarrow Q) \Rightarrow (P \wedge R \Rightarrow Q \wedge R)$$

$$(iii) \quad R \wedge (P \Rightarrow Q) \equiv (\neg R \vee P \Rightarrow R \wedge Q)$$

$$(iv) \quad (P \wedge Q \Rightarrow R) \equiv (P \Rightarrow \neg Q \vee R) .$$

(Einde Opgave 8.)

\* \* \*

Een heel andere manier om uit bestaande predicaten nieuwe te maken is door middel van "universele quantificatie" en "existentiële quantificatie".

Zijn  $P$  en  $Q$  predicaten, dan ook de expressies

$$(\underline{A} x: P: Q) \quad \text{en}$$

$$(\underline{E} x: P: Q) .$$

Het symbool  $\underline{A}$  is de universele quantor en het symbool  $\underline{E}$  de existentiële quantor. De naam  $x$  is de "dummy",  $P$  het "domein" en  $Q$  de "term" van de quantificatie.

Er geldt

$$(39) \quad \neg(\underline{A} x: P: Q) \equiv (\underline{E} x: P: \neg Q)$$

$$(40) \quad \neg(\underline{E} x: P: Q) \equiv (\underline{A} x: P: \neg Q) ,$$

welke regels bekend zijn als de (gegeneraliseerde) wetten van de Morgan .

Ten gevolge van deze wetten komen alle formules over quantificatie in paren, en we zullen ze dan ook paarsgewijs presenteren.

Er geldt

$$(41a) \quad (\underline{A} x: \text{false}: P)$$

$$(41b) \quad \neg(\underline{E} x: \text{false}: P)$$

$$(42a) \quad (\underline{A} x: P \wedge Q: R) \equiv (\underline{A} x: P: \neg Q \vee R)$$

$$(42b) \quad (\underline{E} x: P \wedge Q: R) \equiv (\underline{E} x: P: Q \wedge R)$$

$$(43a) \quad (\underline{A} x: P: Q) \Rightarrow (\underline{A} x: P: Q \vee R)$$

$$(43b) \quad (\underline{E} x: P: Q) \Rightarrow (\underline{E} x: P: Q \vee R) .$$

De regels (41) zeggen dat universele quantificatie over een leeg domein true is en dat existentiële quantificatie over een leeg domein false is.

De regels (42) geven aan hoe tussen domein en term van een quantificatie verhuizing mogelijk is.

De regels (43) leveren de mogelijkheid tot verzwakking van een quantificatie door verzwakking van de term.

We illustreren hoe de eerste der regels

$$(44a) \quad (\underline{A} x: P: \text{true})$$

$$(44b) \quad \neg(\underline{E} x: P: \text{false})$$

volgt uit het voorgaande:

$$\begin{aligned} & (\underline{A} x: P: \text{true}) \\ = & \{ \text{true} \equiv \text{true} \vee \text{true} , (42a) \} \\ & (\underline{A} x: P \wedge \neg \text{true}: \text{true}) \\ = & \{ P \wedge \neg \text{true} \equiv \text{false} \} \\ & (\underline{A} x: \text{false}: \text{true}) \\ = & \{ (41a) \} \\ & \text{true} . \end{aligned}$$

Opgave 9 Laat zien dat

$$\begin{aligned} & (\underline{A} x: P: Q) \equiv (\underline{A} x: P: P \wedge Q) \\ \text{en} \quad & (\underline{E} x: P: Q) \equiv (\underline{E} x: P: \neg P \vee Q) . \end{aligned}$$

(Einde Opgave 9.)

Opgave 10 Toon door berekening aan dat

$$(i) \quad (\underline{A} x: P \wedge Q: P)$$

$$(ii) \quad (\underline{A} x: P: P \vee Q)$$

$$(iii) \quad \text{Uit de geldigheid van } P \Rightarrow Q \text{ volgt de geldigheid van } (\underline{A} x: P: Q)$$

$$(iv) \quad (\underline{E} x: P: \neg P) \equiv \text{false}$$

$$(v) \quad (\underline{E} x: P: Q \wedge R) \Rightarrow (\underline{E} x: P: Q)$$

$$(vi) \quad (\underline{E} x: P: Q \wedge R) \Rightarrow (\underline{E} x: P: Q) \wedge (\underline{E} x: P: R)$$

$$(vii) \quad (\underline{A} x: P: Q) \vee (\underline{A} x: P: R) \Rightarrow (\underline{A} x: P: Q \vee R)$$

$$(viii) \quad (\underline{A} x: P: \neg Q) \equiv (\underline{A} x: Q: \neg P)$$

$$(ix) \quad (\underline{E} x: P: Q) \equiv (\underline{E} x: Q: P) .$$

(Einde Opgave 10.)

Opgave 11 Laat zien dat het onmogelijk is om met de tot dusver gegeven regels een universeel gequantificeerde expressie tot false te herleiden.

(Einde Opgave 11.)

\* \* \*

Voordat we de belangrijkste der resterende rekenregels geven, bespreken we eerst enkele notationale kwesties.

- Is  $R$  een expressie,  $x$  een variabele en  $E$  een expressie van een bij  $x$  passend type, dan is  $R_E^x$  de expressie die uit  $R$  verkregen wordt door in  $R$  elk voorkomen van  $x$  te vervangen door  $E$ .

Bijvoorbeeld geldt, aldus,

$$(x + y)_{x-y}^x = x - y + y \quad \text{en} \quad R_x^x \equiv R \quad \text{en}$$

$$(x = E)_E^x \equiv (E = E_E^x) \quad \text{en} \quad (P_y^x)_x^y \equiv P_x^y .$$

- Is  $R$  een gequantificeerde expressie dan geldt (per definitie) dat de naam van de dummy niet in  $R$  voorkomt. Hieraan komen wij tegemoet door af te spreken dat wij bij de introductie van een gequantificeerde expressie voor de dummy een kersverse naam kiezen, i.e. een naam die nog nergens anders voorkwam.

Bijvoorbeeld geldt, aldus, voor elke (kers)verse  $y$

$$(45a) \quad (\underline{A} \ x: P: Q) \equiv (\underline{A} \ y: P_y^x: Q_y^x) \quad \text{en}$$

$$(45b) \quad (\underline{E} \ x: P: Q) \equiv (\underline{E} \ y: P_y^x: Q_y^x) ,$$

bekend als de regels voor "herbenoeming van de dummy".

Een van de redenen waarom wij om gequantificeerde expressies steeds haakjes zetten is om te benadrukken dat "bij het openingshaakje" de vigerende collectie namen met een naam wordt uitgebreid, namelijk met die van de dummy, en dat "bij het corresponderende sluit-haakje" deze naam weer uit de collectie verdwijnt.

We weiden over de kwestie van de (tekstuele) reikwijdte van namen nu niet verder uit. In een later stadium komen we er nog op terug zodra de structuur van programma's aan de orde wordt gesteld.

\* \* \*

Voor elke  $R$  waar  $x$  niet in voorkomt geldt

$$(46a) \quad (\underline{A} \ x: P: Q) \vee R \equiv (\underline{A} \ x: P: Q \vee R)$$

$$(46b) \quad (\underline{E} \ x: P: Q) \wedge R \equiv (\underline{E} \ x: P: Q \wedge R) ,$$

en voor elk niet-leeg domein, i.e. voor een domein  $P$  waarvoor geldt  $(\underline{E} \ x: P: \text{true})$  ,

$$(47a) \quad (\underline{A} \ x: P: Q) \wedge R \equiv (\underline{A} \ x: P: Q \wedge R)$$

$$(47b) \quad (\underline{E} \ x: P: Q) \vee R \equiv (\underline{E} \ x: P: Q \vee R) .$$

De regels (46) en (47) zijn regels voor distributie van conjunctie en disjunctie over universele en existentiële quantificaties. Let wel dat in (47) een leeg domein niet is toegelaten.

We illustreren de geldigheid van de tweede der "substitutieregels"

$$(48a) \quad (\underline{A} y: x = y: R) \equiv R_x^y$$

$$(48b) \quad (\underline{E} y: x = y: R) \equiv R_x^y \quad :$$

$$\begin{aligned} & (\underline{E} y: x = y: R) \\ = & \{ (x = y) \equiv (x = y) \wedge (x = y), \quad (42b) \} \\ & (\underline{E} y: x = y: x = y \wedge R) \\ = & \{ x = y \wedge R \equiv x = y \wedge R_x^y \} \\ & (\underline{E} y: x = y: x = y \wedge R_x^y) \\ = & \{ (42b) \} \cdot \\ & (\underline{E} y: x = y: R_x^y) \\ = & \{ y \text{ komt niet in } R_x^y \text{ voor; } R_x^y \equiv \text{false} \vee R_x^y; \quad (47b) \} \\ & (\underline{E} y: x = y: \text{false}) \vee R_x^y \\ = & \{ \text{met (44b)} \} \\ & R_x^y . \end{aligned}$$

Met  $E$  een expressie waar  $y$  niet in voorkomt, is een wat algemenere formulering van de substitutieregels gegeven door

$$(49a) \quad (\underline{A} y: y = E: P) \equiv P_E^y$$

$$(49b) \quad (\underline{E} y: y = E: P) \equiv P_E^y \quad .$$

Opgave 12 Ga na hoe (48) uit (49) volgt.

(Einde Opgave 12.)

Merk op dat wij met (48) en (49) in staat zijn een universele quantificatie tot *false* te herleiden en een existentiële quantificatie tot *true*, ingrediënten die nog ontbraken (vergelijk Opgave 11).

Samen met de mogelijkheid om domeinen te splitsen, zoals gegeven door (50), completeert dit onze calculus.

$$(50a) \quad \{\underline{A} x: P \vee Q: R\} \equiv \{\underline{A} x: P: R\} \wedge \{\underline{A} x: Q: R\}$$

$$(50b) \quad \{\underline{E} x: P \vee Q: R\} \equiv \{\underline{E} x: P: R\} \vee \{\underline{E} x: Q: R\}$$

\*       \*       \*

Wij geven, ter afsluiting, nog een aantal voorbeelden.

(i) Voor een niet-leeg domein  $P$  geldt  $\{\underline{E} x: P: P\}$ , immers

$$\begin{aligned} & \{\underline{E} x: P: P\} \\ = & \{\text{elementair}\} \\ & \{\underline{E} x: P: P \wedge \text{true}\} \\ = & \{P \wedge P \equiv P \text{ en (42b)}\} \\ & \{\underline{E} x: P: \text{true}\} \\ = & \{P \text{ is niet leeg}\} \\ & \text{true} . \end{aligned}$$

(ii) Er geldt  $\{\underline{A} x: P: Q\} \wedge \{\underline{A} x: P: R\} \equiv \{\underline{A} x: P: Q \wedge R\}$ , immers

$$\begin{aligned} & \{\underline{A} x: P: Q\} \wedge \{\underline{A} x: P: R\} \\ = & \{\text{met (42a) of met Opgave 10 (viii)}\} \\ & \{\underline{A} x: \neg Q: \neg P\} \wedge \{\underline{A} x: \neg R: \neg P\} \\ = & \{(50a)\} \\ & \{\underline{A} x: \neg Q \vee \neg R: \neg P\} \\ = & \{\text{met (42a)}\} \\ & \{\underline{A} x: P: Q \wedge R\} . \end{aligned}$$

(iii) Met  $x$  en  $y$  uit de gehele getallen geldt

$\{\underline{A} x: x = 2 \cdot y \wedge y > 0: 4 \mid x\} \equiv \{y \leq 0 \vee 2 \mid y\}$ , immers

$$\begin{aligned} & \{\underline{A} x: x = 2 \cdot y \wedge y > 0: 4 \mid x\} \\ = & \{(42a)\} \\ & \{\underline{A} x: x = 2 \cdot y: y \leq 0 \vee 4 \mid x\} \\ = & \{(49a)\} \end{aligned}$$

$$\begin{aligned} & (y \leq 0 \vee 4 \mid x)^x_{2 \cdot y} \\ = & \{ \text{aritmetiek} \} \\ & y \leq 0 \vee 2 \mid y \end{aligned}$$

Opgave 13 Toon door berekening aan dat voor een niet-leeg domein  $P$  geldt

- (i)  $\neg(\underline{\forall} x: P: \neg P)$
- (ii)  $(\underline{\exists} x: P: P)$
- (iii)  $(\underline{\forall} x: P: Q) \Rightarrow (\underline{\exists} x: P: Q)$

(Einde Opgave 13.)

Opgave 14 Toon aan dat

- (i)  $(\underline{\exists} x: P: Q) \vee (\underline{\exists} x: P: R) \equiv (\underline{\exists} x: P: Q \vee R)$
- (ii)  $(\underline{\forall} x: P: Q \vee R) \Rightarrow (\underline{\exists} x: P: Q) \vee (\underline{\forall} x: P: R)$

(Einde Opgave 14.)

Opgave 15 Zij  $P(i: i \geq 0)$  een rij predicaten. Toon aan dat voor elk natuurlijk getal  $n$  (i.e.  $n \geq 0$ ) geldt

- (i)  $(\underline{\forall} i: 0 \leq i < n: P(i))$  voor  $n = 0$ , en  
 $(\underline{\forall} i: 0 \leq i < n + 1: P(i)) \equiv$   
 $(\underline{\forall} i: 0 \leq i < n: P(i)) \wedge P(n)$
- (ii)  $\neg(\underline{\exists} i: 0 \leq i < n: P(i))$  voor  $n = 0$ , en  
 $(\underline{\exists} i: 0 \leq i < n + 1: P(i)) \equiv$   
 $(\underline{\exists} i: 0 \leq i < n: P(i)) \vee P(n)$

(Einde Opgave 15.)

Opgave 16 Toon aan dat voor elke rij  $X(i: 0 \leq i < N)$  van gehele getallen geldt

$$(\underline{A} i: 0 \leq i < N: (\underline{A} j: 0 \leq j < N: X(i) \cdot X(j) \geq 0))$$

$$\equiv (\underline{A} i: 0 \leq i < N: X(i) \geq 0) \vee (\underline{A} j: 0 \leq j < N: X(j) \leq 0)$$

(Einde Opgave 16.)

(Einde 0.0 Predicatenrekening.)

## 0.1 Volledige inductie

Wij spreken af dat wij met getallen gehele getallen bedoelen en met natuurlijke getallen getallen die ten minste 0 zijn.

Voor  $P(i: i \geq 0)$  een rij predicaten noteren we het  $i^{\text{de}}$  element als  $P(i)$  of ook wel als  $P_i$ .

Volledige inductie refereert aan een specifieke methode voor het bewijzen van expressies van de vorm  $(\underline{A} n: n \geq 0: P_n)$ . In plaats van het bewijzen van  $P_n$  voor willekeurige natuurlijke  $n$  kan men volstaan met het aantonen van de zwakkere expressie  $P_n \vee (\underline{E} i: 0 \leq i < n: \neg P_i)$ .

Preciezer, er geldt

$$(0) \quad (\underline{A} n: n \geq 0: P_n)$$

$\equiv$

$$(\underline{A} n: n \geq 0: P_n \vee (\underline{E} i: 0 \leq i < n: \neg P_i)),$$

bekend als "het postulaat der volledige inductie".

Heel uitdrukkelijk vermelden we dat het (voorlopig) essentieel is dat het domein van de universele quantificatie dat der natuurlijke getallen is.

De meest gebruikelijke formulering van het postulaat der volledige inductie is

$$(1) \quad (\underline{A} \ n: n \geq 0: P_n)$$

$$\equiv$$

$$P_0 \wedge (\underline{A} \ n: n \geq 1: P_n \vee \neg P(n-1)) .$$

(Het bewijs van  $P_0$  heet meestal "de basis van de inductie" en het bewijs van  $P_n \vee \neg P(n-1)$  meestal "de stap van de inductie".)

Wij vermelden dat de beide inductieprincipes (gelukkig) dezelfde zijn. In het algemeen geniet formulering (0) de voorkeur boven (1), omdat de expressie  $P_n \vee (\underline{E} \ i: 0 \leq i < n: \neg P_i)$  voor  $n \geq 1$  zwakker is dan de expressie  $P_n \vee \neg P(n-1)$ , en derhalve minstens zo gemakkelijk te bewijzen.

Opgave 0 Laat zien dat de geldigheid van (1) volgt uit die van (0).

(Einde Opgave 0.)

Opgave 1 Laat zien dat elk natuurlijk getal ten minste 2 het product van priemgetallen is.

(Einde Opgave 1.)

De beslissing om  $(\underline{A} \ n: n \geq 0: P_n)$  met volledige inductie te bewijzen is idempotent, i.e. definiëren we voor  $n \geq 0$

$$Q_n: \quad P_n \vee (\underline{E} \ i: 0 \leq i < n: \neg P_i)$$

$$R_n: \quad Q_n \vee (\underline{E} \ i: 0 \leq i < n: \neg Q_i) ,$$

dan volgt  $(\underline{A} \ n: n \geq 0: Q_n \equiv R_n)$ , zodat het besluit om  $(\underline{A} \ n: n \geq 0: Q_n)$  met volledige inductie te bewijzen een leeg besluit is.

Bewijs Voor elke  $n \geq 0$  geldt

$R_n$   
 = {definitie  $R_n$ }  
 $Q_n \vee (\exists i: 0 \leq i < n: \neg Q_i)$   
 = {definitie  $Q_n$ }  
 $P_n \vee (\exists i: 0 \leq i < n: \neg P_i) \vee (\exists i: 0 \leq i < n: \neg Q_i)$   
 = {predicatenrekening (zie Opgave 0.0.14)}  
 $P_n \vee (\exists i: 0 \leq i < n: \neg P_i \vee \neg Q_i)$   
 = {wegens Noot hierna}  
 $P_n \vee (\exists i: 0 \leq i < n: \neg P_i)$   
 = {definitie  $Q_n$ }  
 $Q_n$  .

Noot Voor elke  $i \geq 0$  geldt

true  
 = {definitie  $Q_i$ }  
 $(P_i \Rightarrow Q_i)$   
 = {predicatenrekening}  
 $(P_i \wedge Q_i \equiv P_i)$   
 = {predicatenrekening}  
 $(\neg P_i \vee \neg Q_i \equiv \neg P_i)$   
 (Einde Noot.)  
 (Einde Bewijs.)

Opgave 2 Voor elke deelverzameling  $S$  van de natuurlijke getallen geldt

$$(\exists x: x \geq 0: x \text{ in } S) \\ \equiv \\ (\exists x: x \geq 0: x \text{ in } S \wedge (\forall y: 0 \leq y < x: \neg(y \text{ in } S))) ,$$

klassiekelyk verwoord als "elke niet-lege deelverzameling van de natuurlijke getallen heeft een kleinste element".

Laat zien dat dit postulaat gelijk is aan het postulaat van volledige inductie.

(Einde Opgave 2.)

(Einde 0.1 Volledige inductie.)

## 0.2 Overige begrippen

In deze paragraaf bespreken we een aantal begrippen die we veelvuldig zullen gebruiken bij het programmeren en die wat betreft hun notatie of gebruik afwijken van wat gangbaar is.

### 0.2.0 Algemeen

De symbolen  $=$ ,  $\neq$ ,  $\geq$ ,  $\leq$ ,  $>$  en  $<$  spreken wij uit als "is gelijk aan", "verschilt van", "ten minste", "ten hoogste", "groter dan" en "kleiner dan", respectievelijk.

De natuurlijke getallen zijn de (gehele) getallen die ten minste 0 zijn.

Komt  $x$  uit een domein van  $N$ ,  $N \geq 0$ , consecutieve getallen dat "bij  $k$  begint", dan noteren wij dit meestal als  $k \leq x < k + N$ .

Het lege traject natuurlijke getallen dat bij 0 begint kan zo met louter natuurlijke getallen gekarakteriseerd worden:

$$0 \leq x < 0.$$

Met de notatie  $f(x: P)$  bedoelen we dat  $f$  een functie is van één argument en gedefinieerd is op het domein  $P$ .

De geheeltallige functie  $f(x: p \leq x < q)$ , met  $p \leq q$ , is

- "ascending" betekent  $(\underline{A} x: p + 1 \leq x < q: f(x - 1) \leq f(x))$  ;
- "descending" betekent  $(\underline{A} x: p + 1 \leq x < q: f(x - 1) \geq f(x))$  ;
- "increasing" betekent  $(\underline{A} x: p + 1 \leq x < q: f(x - 1) < f(x))$  ;
- "decreasing" betekent  $(\underline{A} x: p + 1 \leq x < q: f(x - 1) > f(x))$  ;
- "monotoon" betekent dat zij ascending of descending is;
- "constant" betekent dat zij ascending en descending is.

Een functie  $f(x: p \leq x < q)$  noemen wij heel vaak een eindige rij, ter lengte  $q - p$ .

Twee rijen  $f(x: p \leq x < q)$  en  $g(x: p \leq x < q)$ , met  $p \leq q$ , zijn gelijk (notatie:  $f = g$ ) betekent  $(\underline{A} x: p \leq x < q: f(x) = g(x))$ .

Voor twee rijen  $f(x: p \leq x < q)$  en  $g(x: p \leq x < q)$ , met  $p \leq q$ , betekent "f is lexicografisch kleiner dan g" (notatie:  $f < g$ )

$(\underline{E} x: p \leq x < q: f(x) < g(x) \wedge (\underline{A} y: p \leq y < x: f(y) = g(y)))$ .

De lexicografische ordening is een totale ordening, i.e. voor elk tweetal rijen  $f$  en  $g$ , met hetzelfde domein, geldt  $f < g \vee f = g \vee g < f$ .

Het maximum van twee getallen  $x$  en  $y$  noteren wij als  $x \max y$ , en het minimum als  $x \min y$ .

Voor alle  $x$ ,  $y$  en  $z$  geldt:

$(z = x \max y) \equiv (z = x \vee z = y) \wedge z \geq x \wedge z \geq y$ ,

$(z = x \min y) \equiv (z = x \vee z = y) \wedge z \leq x \wedge z \leq y$ .

De operatoren max en min zijn symmetrisch, associatief en

idempotent. Het eenheidselement van max is  $-\text{inf}$ ; het eenheidselement van min is  $+\text{inf}$ .

Soms willen wij een predicaat  $P$  opvatten als een vergelijking in de onbekende  $x$ . Dan noteren we dit als  $x: P$ .

Aldus heeft de vergelijking  $z: z = x \text{ max } y$  voor elke  $x$  en  $y$  precies één oplossing.

De vergelijking  $x: z = x \text{ max } y$  heeft geen enkele oplossing voor  $y > z$  en precies één oplossing voor  $y < z$ .

De zwakste oplossing van  $X: X \Rightarrow P$  is  $P$ .

(Einde 0.2.0 Algemeen.)

### 0.2.1 Andere gequantificeerde expressies

Is  $P$  een predicaat,  $E$  een expressie van het type integer, en heeft de vergelijking

$$x: (P \wedge E \neq 0)$$

een eindig aantal oplossingen, dan is ook

$$(\underline{S} x: P: E)$$

een expressie van het type integer.

Het symbool  $\underline{S}$  is de sommator (sommatie-quantor);  $x$  is de dummy van de gequantificeerde expressie,  $P$  het domein en  $E$  de term.

We geven een aantal rekenregels. Er geldt

$$(0) \quad (\underline{S} x: \text{false}: E) = 0$$

$$(1) \quad (\underline{S} x: x = y: E) = E_y^x$$

$$(2) \quad (\underline{S} x: P: E) + (\underline{S} x: Q: E) \\ = (\underline{S} x: P \vee Q: E) + (\underline{S} x: P \wedge Q: E)$$

$$(3) \quad (\underline{S} x: P: E_0) + (\underline{S} x: P: E_1) \\ = (\underline{S} x: P: E_0 + E_1)$$

(4)  $EO \cdot (\underline{S} x: P: E1) = (\underline{S} x: P: EO \cdot E1)$  mits  $x$  niet in  $EO$  voorkomt.

(5) Voor een verse  $y$   
 $(\underline{S} x: P: E) = (\underline{S} y: P_y^x: E_y^x) .$

De regels spreken voor zich.

Aan de eis dat het aantal oplossingen van  $x: (P \wedge E \neq 0)$  eindig is voldoen we meestal door het domein  $P$  van de sommatie eindig te kiezen.

\*       \*       \*

Het aantal oplossingen van  $x: P \wedge Q$  is, mits dit eindig is, per definitie gelijk aan de "numeriek gequantificeerde expressie"

$$(\underline{N} x: P: Q) .$$

Het symbool  $\underline{N}$  is de "numerieke quantor" (de  $\underline{N}$  is van "the number of").

Er geldt

$$(6) \quad (\underline{N} x: P: Q) = (\underline{N} x: P \wedge Q: \text{true}) .$$

Voor het overige volgen de rekenregels uit die voor de sommatie, aangezien per definitie geldt

$$(7) \quad (\underline{N} x: P: \text{true}) = (\underline{S} x: P: 1) .$$

Universele en existentiële quantificatie kunnen voor eindige domeinen worden geformuleerd in termen van numerieke quantificatie:

$$(8) \quad (\underline{A} x: P: Q) \equiv ((\underline{N} x: P: \neg Q) = 0)$$

$$(9) \quad (\underline{E} x: P: Q) \equiv ((\underline{N} x: P: Q) \geq 1) .$$

De geldigheid van de wetten van de Morgan zou dan volgen uit de geldigheid van

$$((\underline{N} x: P: Q) = 0) \equiv ((\underline{N} x: P: Q) < 1) ,$$

een heuse trivialiteit.

\*       \*       \*

Is de geheeltallige functie  $f(x)$  gedefinieerd op een eindig domein  $P$  dan noteren we het maximum van  $f$  op  $P$  als

$$(\underline{MAX} x: P: f(x))$$

en het minimum als

$$(\underline{MIN} x: P: f(x)) .$$

Er geldt

$$(10) \quad (\underline{MAX} x: \text{false}: f(x)) = -\text{inf}$$

$$(11) \quad (\underline{MAX} x: x = E: f(x)) = f(E) , \text{ mits } x \text{ niet in } E \text{ voorkomt}$$

$$(12) \quad (\underline{MAX} x: P \vee Q: f(x)) \\ = (\underline{MAX} x: P: f(x)) \max (\underline{MAX} x: Q: f(x))$$

$$(13) \quad \text{voor een niet-leeg domein} \\ E + (\underline{MAX} x: P: f(x)) = (\underline{MAX} x: P: E + f(x)) , \\ \text{mits } x \text{ niet in } E \text{ voorkomt.}$$

Het verband tussen  $\underline{MIN}$  en  $\underline{MAX}$  is gegeven door

$$(14) \quad (\underline{MIN} x: P: -f(x)) + (\underline{MAX} x: P: f(x)) = 0 ,$$

en hieruit volgen dan soortgelijke regels voor  $\underline{MIN}$  .

We vermelden nog een definitie van  $\underline{MAX}$  :

$$(15) \quad z = (\underline{MAX} x: P: f(x)) \\ \equiv (\underline{E} y: P: z = f(y)) \wedge (\underline{A} x: P: f(x) \leq z) ,$$

geldig voor niet-leeg domein.

(Einde 0.2.1 Andere gequantificeerde expressies.)

(Einde 0.2 Overige begrippen.)

### 0.3 Gemengde opgaven

(0) Gegeven is een getal  $N$ ,  $N \geq 1$ , en een rij  $X(i: 0 \leq i < N)$  van gehele getallen. Geef de navolgende uitspraken in formule weer.

- (a) Alle waarden in de rij zijn positief.
- (b) Het getal  $p$  komt even vaak voor in  $X(i: 0 \leq i < j)$  als in  $X(i: j \leq i < N)$ .
- (c) Het getal  $r$  is gelijk aan de maximale lengte van enige consecutieve deelrij van  $X(i: 0 \leq i < N)$ , waarvan alle elementen dezelfde waarde hebben.
- (d) De rij bevat geen twee gelijke waarden.
- (e) De rij is een permutatie van de getallen  $0$  tot en met  $N - 1$ .
- (f) De rij  $r(i: 0 \leq i < N)$  bevat de elementen van  $X(i: 0 \leq i < N)$  in opklimmende volgorde.
- (g) Het getal  $k$  is de kleinste index waarvoor  $X(k) = w$ .
- (h)  $r$  is de som van de positieve getallen van de rij.
- (i)  $y$  is het aantal indexparen  $(i, j)$  waarvoor geldt dat de som van en met  $X(0)$  tot en zonder  $X(i)$  gelijk is aan de som van en met  $X(j)$  tot en zonder  $X(N)$ .
- (j)  $r$  is het aantal keren dat het maximum van de rij wordt aangenomen in de rij.

(1) Bepaal voor elke van de hierna genoemde vergelijkingen de zwakste oplossing

- (a)  $X: X \Rightarrow P$
- (b)  $X: X \wedge P \Rightarrow Q$
- (c)  $X: X \vee (P \wedge Q) \Rightarrow Q$
- (d)  $X: X \vee Q \Rightarrow P \vee Q$
- (e)  $X: P \Rightarrow P \vee Q$ .

(2) Bepaal voor elk van de hierna genoemde vergelijkingen de sterkste oplossing

$$(a) \quad X: X \Rightarrow P$$

$$(b) \quad X: P \Rightarrow X$$

$$(c) \quad X: P \Rightarrow X \vee Q .$$

(3) Karakteriseer de oplossingen van

$$(a) \quad X: P \wedge X \equiv Q$$

$$(b) \quad X: (P \vee \neg Q) \wedge X \equiv P \equiv Q .$$

(4) Geef, bij een gegeven getallenrij  $f(x: 0 \leq x < N)$ , een recurrente definitie van  $\{\underline{S} \ x: 0 \leq x < N: f(x)\}$ , en ook van  $\{\underline{MAX} \ x: 0 \leq x < N: f(x)\}$ .

(5) Geef een recurrente definitie voor de lexicografische ordening.

(6) Bewijs door formele manipulatie (en met gebruikmaking van de definitie van lexicografische ordening) dat voor elk tweetal rijen  $f$  en  $g$  met hetzelfde domein

$$\neg(f < g) \wedge \neg(g < f) \equiv (f = g) .$$

(7) Laat met predicaatencalculus zien dat voor elke rij  $P(i: i \geq 0)$  van predicaatencalculus geldt

$$(\underline{A} \ i: i \geq 0: P(i)) \equiv (\underline{A} \ i: i \geq 0: (\underline{A} \ j: 0 \leq j \leq i: P(j))) .$$

(8) In termen van de predicaatenrij  $P(i: i \geq 0)$  definiëren we de predicaatenrij  $Q(j: j \geq 0)$ , in termen waarvan we de predicaatenrij  $R(k: k \geq 0)$  definiëren:

$$Q(0) \equiv P(0), \quad Q(j+1) \equiv Q(j) \vee P(j+1) \quad \text{voor alle } j: j \geq 0;$$

$$R(0) \equiv Q(0), \quad R(k+1) \equiv R(k) \vee Q(k+1) \quad \text{voor alle } k: k \geq 0.$$

Bewijs of weerleg

$$(\underline{A} n: n \geq 0: Q(n) \equiv P(n)) , \quad \text{en}$$

$$(\underline{A} n: n \geq 0: R(n) \equiv Q(n)) .$$

(9) Bewijs met de definities van max en min

$$(a) \quad x + y \geq x \underline{\max} y \equiv x \geq 0 \wedge y \geq 0$$

$$(b) \quad x + (y \underline{\max} z) = (x + y) \underline{\max} (x + z)$$

$$(c) \quad (x \underline{\max} y = x \underline{\min} y) \equiv (x = y) .$$

(10) Laat zien dat voor elke getallenrij  $X(i: 0 \leq i < N)$ ,  $N \geq 1$ , en voor elke gehele  $M$  geldt

$$(\underline{S} i: 0 \leq i < N: X(i)) < M \vee (\underline{E} i: 0 \leq i < N: N \cdot X(i) \geq M) ,$$

het "Pigeon hole principle".

(11) Gegeven zijn  $N$  getallen,  $N \geq 3$ .

Laat zien dat er onder die  $N$  getallen ten minste twee zijn wier som of verschil een veelvoud van  $N$  is.

(12) De rij  $F(i: i \geq 0)$  der Fibonacci-getallen is gedefinieerd met het recurrentie-schema

$$F(0) = 0; \quad F(1) = 1;$$

$$\text{voor alle } n: n \geq 0: F(n+2) = F(n+1) + F(n) .$$

Bewijs dat voor alle  $n: n \geq 0$

$$F(2 \cdot n + 1) = (F(n))^2 + (F(n + 1))^2$$

$$F(2 \cdot n + 2) = 2 \cdot F(n) \cdot F(n + 1) + (F(n + 1))^2$$

(13) Voor een rij  $X(i: 0 \leq i < N)$  definiëren wij  $s$  door

$$s = (\text{MIN } p: 0 \leq p < N \wedge (\text{A } q: p \leq q < N: X(p) \leq X(q)): p) .$$

Bewijs dat  $(\text{A } r: 0 \leq r < s: X(r) > X(s))$  .

(14) Bewijs dat voor elke getallenrij  $X(i: 0 \leq i < N)$  ,  $N \geq 1$  , geldt

$$\begin{aligned} & (\text{A } i: 1 \leq i < N: X(0) < X(i)) \\ \equiv & (\text{A } i: 1 \leq i < N: (\text{E } j: 0 \leq j < i: X(j) < X(i))) . \end{aligned}$$

(15) Bewijs dat voor elke getallenrij  $X(i: 0 \leq i < N)$  ,  $N \geq 1$  , en voor elk getal  $M$  geldt

$$\begin{aligned} (\text{E } i: 0 \leq i < N: X(i) \geq M) & \equiv ((\text{MAX } i: 0 \leq i < N: X(i)) \geq M) \\ \text{en } (\text{A } i: 0 \leq i < N: X(i) \geq M) & \equiv ((\text{MIN } i: 0 \leq i < N: X(i)) \geq M) . \end{aligned}$$

(Einde 0.3 Gemengde opgaven.)

(Einde 0 Algemene inleiding.)

## 1 FUNCTIONELE SPECIFICATIES EN BEWIJSVERPLICHTINGEN

Was de tekst van deze handleiding tot nu toe tamelijk in zichzelf besloten, vanaf hier zal dit niet meer het geval zijn omdat op de collegetekst wordt ingehaakt. Van de lezer wordt bekendheid met de relevante collegestof verondersteld.

\*       \*       \*

We resumeren de belangrijkste begrippen.  
Om der eenvouds wille kiezen we de gehele getallen als het domein van de lokale variabelen.

Voor een statement list  $S$  en voor predicaten  $P$  en  $Q$  betekent de geldigheid van de expressie

$$| [ x: \text{int } \{P\}; S \{Q\} ] |$$

dat uitvoering van  $S$  voor een begintoestand die aan de beginconditie  $P$  voldoet een eindtoestand bewerkstelligt die aan de eindconditie  $Q$  voldoet.

Hiermee is een verband aangestipt tussen expressies van de vorm  $| [ x: \text{int } \{P\}; S \{Q\} ] |$  en een appreciatie van deze expressies als machinaal uitvoerbare code. Deze mechanistische appreciatie zal in wat volgt zo goed als geen enkele rol spelen.

In menig programmeervraagstuk zijn  $P$  en  $Q$  gegeven en kan  $| [ x: \text{int } \{P\}; S \{Q\} ] |$  worden opgevat als een vergelijking in  $S$  : voor sommige  $S$  is het een geldige expressie en voor sommige

(lees: de meeste)  $S$  niet. De redactie van zo'n vraagstuk luidt daarom kortweg

$S: \llbracket x: \text{int } \{P\}; S \{Q\} \rrbracket$  .

\* \* \*

Voor willekeurige predicaten  $P$  en  $Q$ , statement lists  $S$ ,  $S0$  en  $S1$ , Boolese expressies  $B0$  en  $B1$ , en voor willekeurige integer expressie  $E$  geldt per definitie

(0)  $\llbracket x: \text{int } \{P\}; \text{skip } \{Q\} \rrbracket$

betekent

$P \Rightarrow Q$ ,

het "postulaat van de skip".

(1)  $\llbracket x, y: \text{int } \{P\}; x := E \{Q\} \rrbracket$

betekent

$P \Rightarrow Q_E^x$ ,

het "postulaat van de assignment".

(2)  $\llbracket x: \text{int } \{P\}; S0; S1 \{Q\} \rrbracket$

betekent

er bestaat een predicaat  $H$  zo dat

$\llbracket x: \text{int } \{P\}; S0 \{H\} \rrbracket$  en  $\llbracket x: \text{int } \{H\}; S1 \{Q\} \rrbracket$ ,

het "postulaat van de concatenatie".

(3)  $\llbracket x: \text{int}$

$\{P\}$

; if  $B0 \rightarrow S0$

$\parallel B1 \rightarrow S1$

fi

$\{Q\}$

$\rrbracket$

betekent

$P \Rightarrow B0 \vee B1$  en

$[[x: \text{int } \{P \wedge B0\}; S0 \{Q\}]]$  en

$[[x: \text{int } \{P \wedge B1\}; S1 \{Q\}]]$ ,

het "postulaat van de alternatieve statement".

(4)  $[[x: \text{int}$   
 $\{P\}$   
 $;$  do  $B0 \rightarrow S0$   
 $\quad \parallel B1 \rightarrow S1$   
 $\quad$  od  
 $\{Q\}$   
 $]]$

betekent

er bestaat een predicaat  $H$  en een integer functie  $vf$   
 zo dat

$P \Rightarrow H$  en

$[[x: \text{int } \{H \wedge B0 \wedge vf = VF\}; S0 \{H \wedge vf < VF\}]]$  en

$[[x: \text{int } \{H \wedge B1 \wedge vf = VF\}; S1 \{H \wedge vf < VF\}]]$  en

$H \wedge (B0 \vee B1) \Rightarrow vf \geq 0$  en

$H \wedge \neg B0 \wedge \neg B1 \Rightarrow Q$ ,

het "postulaat van de repetitieve statement", ook bekend  
 als "de invariantiestelling voor de repetitieve statement".  
 Het predicaat  $H$  heet de "invariant" van de repetitie en  
 de integer functie  $vf$  de "variante functie".

\* \* \*

Aangaande het postulaat van de concatenatie delen we nog mee dat  
 de zwakste  $P$  die bij een gegeven  $Q$  past verkregen kan worden  
 door de zwakste  $H$  te kiezen waarvoor  $[[x: \text{int } \{H\}; S1 \{Q\}]]$   
 en bij die  $H$  de zwakste  $P$  waarvoor  $[[x: \text{int } \{P\}; S0 \{H\}]]$ .

\* \* \*

Voor het "rekenen" met functionele specificaties vermelden we twee algemene regels, geldig voor elke  $S$  :

- (5)     Uit  
            $[[x: \text{int } \{P_0\}; S \{Q_0\}]]$  en  
 $P_1 \Rightarrow P_0$    en    $Q_0 \Rightarrow Q_1$   
           volgt  
            $[[x: \text{int } \{P_1\}; S \{Q_1\}]]$  .
- (6)     Uit        $[[x: \text{int } \{P_0\}; S \{Q_0\}]]$   
           en        $[[x: \text{int } \{P_1\}; S \{Q_1\}]]$   
           volgen  $[[x: \text{int } \{P_0 \wedge P_1\}; S \{Q_0 \wedge Q_1\}]]$   
           en        $[[x: \text{int } \{P_0 \vee P_1\}; S \{Q_0 \vee Q_1\}]]$  .

Aan (5) refereren we als we zeggen dat in een functionele specificatie de preconditione versterkt mag worden en de postconditie verzwakt. In (6) worden omstandigheden aangegeven waaronder de preconditione verzwakt mag worden en de postconditie versterkt.

\*       \*       \*

## Opgaven

(0) In  $[[x: \text{int } \{P\}; S \{Q\}]]$  mogen voor  $P$  en voor  $Q$  elk der vier predicaten  $x = X$  ,  $x = \text{abs}(X)$  ,  $\text{abs}(x) = X$  en  $\text{abs}(x) = \text{abs}(X)$  gesubstitueerd worden. In welke der gevallen ontstaat de functionele specificatie van een programma  $S$  dat de absolute waarde van een getal berekent?

(1) Bespreek de functionele specificatie

$[[x, y: \text{int } \{x = X\}; S \{y = Y\}]]$  .

(2) Geldt  $\llbracket x: \text{int} \{2 \cdot x \leq X\}; \text{skip} \{x \leq X\} \rrbracket$  ?

(3) Wat betekent  $\llbracket x: \text{int} \{P\}; \text{skip} \{P \wedge Q\} \rrbracket$  voor  $P$  en  $Q$  ?

(4) Toon aan dat

$\llbracket x, y, m: \text{int} \{x \geq 0 \wedge y \geq 0\}; m := x + y \{m \geq x \text{ max } y\} \rrbracket$  .

(5) Bepaal de zwakste  $P$  zo dat

$\llbracket x, y, m: \text{int} \{P\}; m := x \text{ max } y \{m > x + y\} \rrbracket$  .

(6) Bepaal de zwakste  $P$  zo dat voldaan is aan

$\llbracket x, y, z: \text{int}; c, d: \text{bool} \{P\}; S \{Q\} \rrbracket$  , voor elk der onderstaande gevallen

S	Q	S	Q
(a) $x := 3$	$x = 3$	(h) $x := x - y$	$x = x - y$
(b) $x := 3$	$x \neq y$	(i) $x := 2 * x + 1$	$x = 2 * x + 1$
(c) $x := 3$	true	(j) $c := c \wedge d$	$c \equiv c \wedge d$
(d) $x := x + 1$	$y = 7$	(k) $c := c \vee d$	$c \equiv c \vee d$
(e) $x := y / 2$	$x = 3$	(l) $c := c \equiv d$	$c \equiv c \equiv d$
(f) $x := y / 2$	true	(m) $x := y \text{ mod } z$	$0 \leq x < z$
(g) $z := x$	$z = x \text{ min } y$	(n) $z := (x + y) / 2$	$x < z < y$

(7) Geef een statement list  $S$  zo dat voor alle  $P$  voldaan is aan  $\llbracket x, y: \text{int} \{P\}; x := 2 * x + y; S \{P\} \rrbracket$  .

(8) Karakteriseer alle  $P$  waarvoor geldt

$\llbracket x, y: \text{int} \{P\}; x := (x - y) / 2; x := 2 * x + y \{P\} \rrbracket$  .

(9) Geef een paar expressies  $P$  zo dat voldaan is aan  $\llbracket x, y: \text{int } \{P\}; y := x; x := y \{P\} \rrbracket$ , en geef ook een paar expressies die niet voldoen.

(10) Wat betekent de geldigheid van  $\llbracket x, y, z: \text{int } \{P\}; x := y; x := z \{Q\} \rrbracket$  voor het paar  $P, Q$  ?  
Idem:  $\llbracket x, y, z: \text{int } \{P\}; x := z \{Q\} \rrbracket$ .

(11) Toon de equivalentie aan van  $\llbracket x, y: \text{int } \{P\}; x := y; y := x \{Q\} \rrbracket$  en  $\llbracket x, y: \text{int } \{P\}; x := y \{Q\} \rrbracket$ .

(12) Van de vergelijking  $P: \llbracket x: \text{int } \{P\}; x := E \{Q\} \rrbracket$  is, per definitie,  $Q_E^x$  de zwakste oplossing.

Van de vergelijking  $Q: \llbracket x: \text{int } \{P\}; x := E \{Q\} \rrbracket$  is  $\llbracket E_y x = E_y^x P_y^x \rrbracket$  een oplossing. Bewijs dit.

We delen mee dat het de sterkste oplossing is.

We merken op dat wat de assignment statement betreft het berekenen van precondities neerkomt op substitutie en het berekenen van postcondities op het ingewikkelder (en minder vertrouwde) procédé van parametrisering.

Dit puur technische verschil verklaart waarom wij er de voorkeur aan geven programma's "achterwaarts" te behandelen.

(a) Bepaal de sterkste  $Q$  die voldoet aan  $\llbracket x: \text{int } \{x \geq 7\}; x := x - 7 \{Q\} \rrbracket$ , en voor die  $Q$  de zwakste  $P$  die voldoet aan  $\llbracket x: \text{int } \{P\}; x := x - 7 \{Q\} \rrbracket$ .

(b) Bepaal de sterkste  $Q$  die voldoet aan  $\llbracket x: \text{int } \{0 \leq x < 4\}; x := x * x \{Q\} \rrbracket$ , en voor die  $Q$  de zwakste  $P$  die voldoet aan  $\llbracket x: \text{int } \{P\}; x := x * x \{Q\} \rrbracket$ .

(13) Bewijs of weerleg dat voor alle gehele  $x$  en  $y$

$$(a) \quad (x + y) \bmod 7 = x \bmod 7 + y \bmod 7$$

$$(b) \quad (x + y) \bmod 7 = (x \bmod 7 + y \bmod 7) \bmod 7$$

$$(c) \quad y = 0 \text{ cor } (x \bmod y) \bmod y = x \bmod y$$

$$(d) \quad (x + x \bmod 2)^x_x + x \bmod 2 = x + x \bmod 2$$

$$(e) \quad 2 * ((x - x \bmod 2) \text{div } 2) = x - x \bmod 2 .$$

(14) Toon aan dat

```

| [ c, d: bool { (c ≡ C) ∧ (d ≡ D) }
  ; c := (c ≡ d); d := (c ≡ d); c := (c ≡ d)
    { (c ≡ D) ∧ (d ≡ C) }
] | .

```

(15) Toon aan dat

```

| [ c, d: bool { c ≡ C }; c := (c ≡ d); c := (c ≡ d) { c ≡ C } ] | .

```

(16) Toon aan dat

```

| [ x, y, z: int { x · y = z ∧ x mod 2 = 0 }; x := x / 2; y := 2 * y
  { x · y = z } ] | .

```

(17) Bepaal de zwakste Boolese expressie  $B$  waarvoor geldt

```

| [ x, y, z: int { x · y = z ∧ B }; x := (x - 1) / 2; z := z - y;
  y := 2 * y { x · y = z } ] | .

```

(18) Bepaal een Boolese expressie  $B$ , in waarde verschillend van false, zo dat, met  $P$  gedefinieerd als

$n \geq 2 \wedge (\underline{A} d: 2 \leq d < n: \neg d | x)$ , voldaan is aan

$\llbracket n, x: \text{int } \{P \wedge B\}; n := n + 1 \{P\} \rrbracket$  .

Laat zien dat  $(P \wedge n^2 > x) \Rightarrow (\exists d: 2 \leq d < x: \neg d \mid x)$  .

(19) Bewijs dat voor elke  $P$

$\llbracket x: \text{int } \{P\}; \text{if } \text{true} \rightarrow x := 1 \ \square \ \text{true} \rightarrow x := -1 \text{ fi } \{ \text{abs}(x) = 1 \} \rrbracket$  .

(20) Los op

$P: \llbracket x: \text{int } \{P\}; \text{if } \text{true} \rightarrow x := 1 \ \square \ \text{true} \rightarrow x := -1 \text{ fi } \{x = 1\} \rrbracket$  .

(21) Is de statement list

"if  $x = 1 \rightarrow x := -1 \ \square \ x = -1 \rightarrow x := 1$  fi" een oplossing van

$S: \llbracket x: \text{int } \{x = -X\}; S \{x = X \wedge \text{abs}(x) = 1\} \rrbracket$  ?

En van

$S: \llbracket x: \text{int } \{x = -X \wedge \text{abs}(x) = 1\}; S \{x = X\} \rrbracket$  ?

(22) Bewijs dat de expressies

$\llbracket x: \text{int}$   
 $\{P\}$   
 $; \text{if } B0 \rightarrow S0; S2 \ \square \ B1 \rightarrow S1; S2 \text{ fi}$   
 $\{Q\}$

$\rrbracket$

en

$\llbracket x: \text{int}$   
 $\{P\}$   
 $; \text{if } B0 \rightarrow S0 \ \square \ B1 \rightarrow S1 \text{ fi}$   
 $; S2$   
 $\{Q\}$

$\rrbracket$

equivalent zijn voor alle predicaten  $P$  en  $Q$  , alle Boolese expressies  $B0$  en  $B1$  en alle statement lists  $S0$  ,  $S1$  en  $S2$  .

(23) Bewijs dat uit  $\llbracket x: \text{int } \{B\}; S0 \{B\} \rrbracket$  en  
 $\llbracket x: \text{int } \{P\}; \underline{\text{if}} B \rightarrow S0; S1 \underline{\text{fi}} \{Q\} \rrbracket$  volgt dat  
 $\llbracket x: \text{int } \{P\}; S0; \underline{\text{if}} B \rightarrow S1 \underline{\text{fi}} \{Q\} \rrbracket$  .

(24) Onderzoek de equivalentie van

$\llbracket x, y, z: \text{int } \{P\}; \underline{\text{if}} x \geq y \rightarrow S2; S0 \quad \vee \quad y \geq x \rightarrow S2; S1 \underline{\text{fi}} \{Q\} \rrbracket$

en

$\llbracket x, y, z: \text{int } \{P\}; S2; \underline{\text{if}} x \geq y \rightarrow S0 \quad \vee \quad y \geq x \rightarrow S1 \underline{\text{fi}} \{Q\} \rrbracket$

voor

$S2$  achtereenvolgens "skip", " $x, y := x + 1, y + 1$ " en  
" $x, y := y, x$ " .

(25) Bewijs

$\llbracket x, y: \text{int}$   
 $\quad \{y = 2 \cdot x \vee y = -2 \cdot x + 1\}$   
 $\quad ; \underline{\text{if}} y \bmod 2 = 0 \rightarrow x := x - y$   
 $\quad \quad \vee \quad y \bmod 2 = 1 \rightarrow x := x + y$   
 $\quad \underline{\text{fi}}$   
 $\quad ; y := y + 1$   
 $\quad \{y = 2 \cdot x \vee y = -2 \cdot x + 1\}$   
 $\rrbracket$  .

(26) Met  $P$  gedefinieerd als  $x > 0 \wedge y > 0$  wordt gevraagd  
naar Boolese expressies  $B0$  en  $B1$  zo dat

$\llbracket x, y: \text{int}$   
 $\quad \{P\}$   
 $\quad ; \underline{\text{if}} B0 \rightarrow x, y := y, x \quad \vee \quad B1 \rightarrow x := x - y \underline{\text{fi}}$   
 $\quad \{P\}$   
 $\rrbracket$  .

Bepaal eveneens Boolese expressies  $C0$  en  $C1$  zo dat

```

| [ x, y: int
  { P  $\wedge$  x + 2 * y = 0 }
  ; if C0  $\rightarrow$  x, y := y, x  $\square$  C1  $\rightarrow$  x := x - y fi
  { P  $\wedge$  x + 2 * y < 0 }
| ] .

```

Leid af dat

```

| [ x, y: int
  { P }
  ; do x < y  $\rightarrow$  x, y := y, x  $\square$  x > y  $\rightarrow$  x := x - y od
  { P  $\wedge$  x = y }
| ] .

```

(27) Lettend op de postulaten voor de alternatieve en de repetitieve statement zijn wij, voor gegeven predicaten  $P$ ,  $Q$  en  $R$  en voor een gegeven statement list  $S$ , regelmatig uitgedaagd om uitgaande van de geldigheid van  $| [ x: \text{int } \{Q\}; S \{R\} ] |$  een niet al te sterke Boolese expressie  $B$  te vinden zo dat  $| [ x: \text{int } \{P \wedge B\}; S \{R\} ] |$  volgt.

Daartoe dienen wij oplossingen van de vergelijking

$X: P \wedge X \Rightarrow Q$  te beschouwen en wel in het bijzonder die oplossingen waarvan de tekstuele verschijningsvorm een Boolese expressie is. Ga dit na.

Is een oplossing niet (of niet eenvoudig genoeg) als Boolese expressie te representeren dan kan men proberen zo'n oplossing te "vereenvoudigen":

als  $P \wedge Y$  een oplossing is, dan ook de "eenvoudiger" expressie  $Y$ ;

als  $\neg P \vee Y$  een oplossing is, dan ook de "eenvoudiger" expressie  $Y$ .

Vind een Boolese expressie  $B$ , verschillend van  $\text{false}$ , zo dat  $P \wedge B \Rightarrow Q$  voor

$P \equiv (\exists k: k \geq 0: x = 2^k)$  en  $Q \equiv (\exists k: k \geq 1: x = 2^k)$ .

Ook voor

$P \equiv (r = (\underline{N} \ i: 0 \leq i < n: X(i) = 0))$  en

$Q \equiv (r = (\underline{N} \ i: 0 \leq i < n + 1: X(i) = 0))$ .

(28) Bereken Boolese expressies  $B0$ ,  $B1$  en  $B2$  zo dat

```

| [ x: int
    {x = X}
    ; if  $B0 \rightarrow x := -x$  []  $B1 \rightarrow \text{skip}$  []  $B2 \rightarrow x := 0$  fi
    {x = abs(X)}
| ] .

```

(29) Bepaal Boolese expressies  $B0$  en  $B1$  zo dat

```

| [ x, y, z: int
    { $x^y \cdot z = C \wedge x \geq 1 \wedge y \geq 1$ }
    ; if  $B0 \rightarrow x, y := x * x, y / 2$ 
      []  $B1 \rightarrow x, y, z := x * x, (y - 1) / 2, z * x$ 
    fi
    { $x^y \cdot z = C \wedge x \geq 1 \wedge y \geq 0$ }
| ] .

```

(30) Bewijs dat

$| [ x, y: \text{int } \{y > 0\}; \underline{\text{do}} \ x \leq 0 \rightarrow x := x + y \underline{\text{od}} \ \{x > 0\} ] |$ .

(31) Bewijs dat

```

| [ x, y: int {true}; do  $x < y \rightarrow x, y := y, x$  od  $\{x \geq y\}$  ] | ,
en dat
| [ x, y, z: int
    {true}

```

```

; do  $x < y \rightarrow x, y := y, x \parallel y < z \rightarrow y, z := z, y$  od
  { $x \geq y \wedge y \geq z$ }
]I .

```

(32) Geldt

```

I[  $x, y: \text{int}$ 
  {true}
; do  $x < y \rightarrow x, y := y, x \parallel y < x \rightarrow y, x := x, y$  od
  { $x = y$ }
]I ?

```

(33) Zijn

```

I[  $x: \text{int } \{P\}; \text{ do } B0 \rightarrow S \parallel B1 \rightarrow S \text{ od } \{Q\} ]I
\text{ en }
I[  $x: \text{int } \{P\}; \text{ do } B0 \vee B1 \rightarrow S \text{ od } \{Q\} ]I
\text{ equivalent?}$$ 
```

(34) Zijn

```

I[  $x: \text{int } \{P\}; \text{ do } B0 \rightarrow S0 \parallel B1 \rightarrow S1 \text{ od } \{Q\} ]I
\text{ en }
I[  $x: \text{int } \{P\}; \text{ do } B0 \rightarrow S0 \parallel B1 \wedge \neg B0 \rightarrow S1 \text{ od } \{Q\} ]I
\text{ equivalent?}$$ 
```

(35) Uit  $P \Rightarrow (B0 \vee B1)$

```

en      I[  $x: \text{int } \{P \wedge B0\}; S0 \{P\} ]I
en      I[  $x: \text{int } \{P \wedge B1\}; S1 \{P\} ]I
volgt   I[  $x: \text{int }
          \{P\}$$$ 
```

```

; do B0  $\rightarrow$  S0  $\square$  B1  $\rightarrow$  S1 od
  {false}

```

] | , mits de repetitie eindigt.

Ga dit na.

Welke conclusie valt hieruit te trekken?

Opmerking De conclusie, het "postulaat van de uitgesloten beëindiging" behelzende, komen we in een heel ander verband tegen als het "postulaat van de uitgesloten dodelijke omarming". Dodelijke omarming (deadly embrace; deadlock) betreft het "verschijnsel" waarbij een aantal samenwerkende (reken)processen zich in een zodanige toestand heeft gemaneuvreerd dat voor elk der processen voortgang onmogelijk is.

(Einde Opmerking.)

(36) Bepaal de zwakste P zo dat

```

|[ x: int {P}; do x  $\neq$  0  $\rightarrow$  x := x - 1 od {x = 0} ]| .

```

Bepaal tevens de zwakste P zo dat

```

|[ x: int {P}; do x  $\neq$  0  $\rightarrow$  x := x - 2 od {x = 0} ]| .

```

Bestudeer

```

|[ x: int {P}; do x  $\neq$  0  $\rightarrow$  x := x - 1
                 $\square$  x  $\neq$  0  $\rightarrow$  x := x - 2
                od
                {x = 0}

```

] | .

(37) Bewijs dat voor een Boolese functie  $B(y: y \geq 0)$  geldt

```

|[ x: int
  {(E y: y  $\geq$  0: B(y))}
  ; x := 0; do  $\neg B(x)$   $\rightarrow$  x := x + 1 od
  {B(x)  $\wedge$  (A z: 0  $\leq$  z < x:  $\neg B(z)$ )}
]| ,

```

de "Linear Search Theorem".

Laat zien hoe met bovenstaand programma volgt dat

$$(\exists y: y \geq 0: B(y))$$

$$\equiv (\exists y: y \geq 0: B(y) \wedge (\forall z: 0 \leq z < y: \neg B(z))) ,$$

het postulaat van volledige inductie.

We vermelden dat de postulaten van volledige inductie en van de repetitieve statement voor berekenbare Boolese functies hetzelfde behelzen.

(Let wel: niet alle functies zijn berekenbaar: voor een getallenrij  $x(i: i \geq 0)$  baart de berekening van een  $n$  zodat  $n \geq 0 \wedge (\forall i: n \leq i: x(i) \geq 7)$  grote zorgen, zelfs al is de existentie van zo'n  $n$  bekend.)

(38) Zij  $f(x)$  voor elke gehele  $x$  een toegelaten integer expressie en zij  $f$  bovendien increasing.

Bepaal de zwakste  $P$  zo dat

$[[ x: \text{int } \{P\}; \text{do } f(x) > x \rightarrow x := f(x) \text{ od } \{f(x) \leq x\} ]]$  .

(39) Bepaal de zwakste  $P$  zo dat

$[[ x, y, z: \text{int}$

$\{P\}$

$; \text{do } x < y \rightarrow x := x + 1$

$\quad \text{od } y < z \rightarrow y := y + 1$

$\quad \text{od } z < x \rightarrow z := z + 1$

$\text{od}$

$\{x = y \wedge y = z \wedge z = x\}$

$]]$  .

(40) Bewijs dat

```

| [ x, a, b, u, v: int
  {  $a \geq 1 \wedge b \geq 1 \wedge u \geq 0 \wedge v \geq 0 \wedge x = u \cdot a - v \cdot b$  }
; do  $x > 0 \rightarrow x, u := x - a, u - 1$ 
   $\square x < 0 \rightarrow x, v := x + b, v - 1$ 
od
  {  $x = 0$  }
]| .

```

(41) Bewijs dat

```

| [ x, y: int
  {true}
; do  $x > y$ 
   $\rightarrow$  if  $x \bmod 2 = 0 \vee y \bmod 2 = 0 \rightarrow x, y := x - 1, y + 1$ 
     $\square x \bmod 2 = 1 \vee y \bmod 2 = 1 \rightarrow$ 
       $x, y := x + x \bmod 2, y + y \bmod 2$ 
  fi
od
  {  $x \leq y$  }
]| .

```

(42) Laat zien dat het binnenblok, met als functionele specificatie,

```

| [ y: int
  {  $x \geq 0$  }
; statement list
  {  $y^2 \leq x < (y + 1)^2$  }
;  $b := (x = y * y)$ 
]|

```

voldoet aan

S:  $| [ x: \text{int}; b: \text{bool} \{x \geq 0\}; S \{b \equiv (\exists z: z \geq 0: x = z^2)\} ] | .$

(43) Welke van de onderstaande teksten zijn correcte functionele specificaties van een programma ter berekening van de kleinste  $N$  derde machten van natuurlijke getallen?

- (a)     [[ N: int; f(x:  $0 \leq x < N$ ): array of int  
           {N  $\geq 0$ }  
           ; cubes  
           {(A x:  $0 \leq x < N$ : f(x) =  $x^3$ )}  
       ]] .
- (b)     [[ N: int; f(x:  $0 \leq x < N$ ): array of int  
           {N = NO  $\wedge$  N  $\geq 0$ }  
           ; cubes  
           {N = NO  $\wedge$  (A x:  $0 \leq x < N$ : f(x) =  $x^3$ )}  
       ]] .
- (c)     [[ N: int  
           {N  $\geq 0$ }  
           ; [[ f(x:  $0 \leq x < N$ ): array of int  
               ; cubes  
               {(A x:  $0 \leq x < N$ : f(x) =  $x^3$ )}  
           ]]  
       ]] .

(44) Geef een functionele specificatie van een programma ter bepaling van de som der cijfers uit de decimale representatie van een natuurlijk getal.

(Einde Opgaven.)

(Einde 1 Functionele specificaties en bewijsverplichtingen.)

## 2 PROGRAMMEEROPGAVEN

Dit hoofdstuk, dat de kern van deze handleiding vormt, bestaat in hoofdzaak uit een lange reeks programmeeropgaven.

Voordat wij hiermee van start gaan is het wenselijk eerst even in te gaan op de volgorde der vraagstukken, op hun redactie en op een mogelijke stijl van presentatie van de oplossingen.

De vraagstukken zijn een heel klein beetje gerangschikt naar opklimmende moeilijkheid. Dit betekent dat de serie begint met opgaven waarvan min of meer vaststaat dat ze tot de eenvoudige opgaven behoren en dat de wat ingewikkelder vraagstukken wat verderop staan. Dit betekent ook dat de moeilijkheidsgraad, de serie volgend, danig kan fluctueren. De lezer zij hierop voorbereid.

Elke programmeeropgave is uiteindelijk van de vorm: "Geef voor gegeven predicaten  $P$  en  $Q$  een statement list die een oplossing is van de vergelijking  $S: \llbracket x: \text{int } \{P\}; S \{Q\} \rrbracket .$ " .

In die vraagstukken waar pre- en postconditie heel expliciet gegeven zijn volstaan we met het opgeven van de vergelijking.

In menig vraagstuk geven wij "slechts" een (verbale) karakterisering van het gewenste netto-effect van het te construeren programma. In zulke gevallen behoort de expliciete formulering van pre- en postconditie eveneens tot de taak van de lezer: het opstellen van functionele specificaties is onlosmakelijk met programmeren verbonden.

Ter vermindering van de hoeveelheid schrijfwerk, ter registratie van bewijsverplichtingen en ten behoeve van een compacte presentatie van programma's voeren we het afkortingsmechanisme in dat bekend staat onder de naam "programma-annotatie". We illustreren het aan een voorbeeld.

```

| [ N: int
    {N ≥ 0}
; | [ a: int
    ; | [ b: int
        ; a, b := 0, N + 1
        {invariant P: 0 ≤ a < b ∧ a2 ≤ N < b2, zie Noot0.
        variante functie: b - a
        }
        ; do a + 1 ≠ b
            → | [ c: int
                ; c := (a + b) div 2
                {P ∧ a < c < b, zie Noot1}
                ; if c * c ≤ N → a := c {P, zie Noot2}
                    [] N < c * c → b := c {P, als Noot2}
                fi
                {P}
            ] |
        ] |
    od
    {P ∧ a + 1 = b}
] |
{dus, a2 ≤ N < (a + 1)2}
] |
] | .

```

Bovenstaand programma is vrij volledig geannoteerd. De volledigheid bestaat hierin dat voor elke constituerende statement zowel pre- als postconditie uit de tekst zijn af te lezen.

Aldus registreert de geannoteerde programmatekst de bewijslast: voor elk der voorkomende statements dient het opgegeven paar "randcondities" te volgen uit het bij het statement behorende postulaat.

We vestigen er de aandacht op dat elk der zo ontstane bewijsverplichtingen nagekomen dient te worden.

Tegelijkertijd adopteren we de conventie om voor die bewijsverplichtingen die uitdrukkelijk nadere aandacht vergen in de postconditie een hint toe te voegen.

In ons voorbeeld komen vijf van zulke hints voor, drie in de vorm van een verwijzing naar een Noot, een ("als Noot2") die een indicatie geeft over de vorm van een bewijs, en een in de vorm van een (erg overbodige) "dus".

Op grond van onze geannoteerde tekst dient derhalve toegevoegd te worden

Noot0, behelzend het bewijs van:  $N \geq 0 \Rightarrow P_{0,N+1}^{a,b}$  ;

Noot1, behelzend het bewijs van:

$$P \wedge a + 1 \neq b \Rightarrow (P \wedge a < c < b)_{(a+b)\text{div } 2}^c ;$$

Noot2, behelzend het bewijs van

$$P \wedge a < c < b \wedge c^2 \leq N \Rightarrow P_c^a ,$$

waarmee wij het programma uit het voorbeeld voldoende "gedocumenteerd" achten.

Beëindiging van repetitieve statements behandelen we steeds door opgave van een variante functie. In heel veel gevallen is het beëindigingsbewijs triviaal (we zullen later zien waarom) en zullen we --net als in ons voorbeeld-- volstaan met de vermelding van de gekozen variante functie. In de minder triviale gevallen mag het beëindigingsbewijs nimmer ontbreken.

Opmerking Oplettende lezers zal het niet ontgaan zijn dat het met onze conventie betreffende geannoteerde programmateksten mogelijk is de vermelding van enig bewijs te onderdrukken door "hintloos" te annoteren.

De mogelijkheid tot hintloos annoteren is evenwel opzettelijk gekozen voor de werkelijk triviale programma's.

Het is --en wij dienen dat te vermelden-- een der kenmerken van de allerbeste programmeurs dat zij niet te snel iets als vanzelfsprekend beschouwen. Zij weten dat een gebrek aan besef van wat wel en wat niet triviaal is onverwijld leidt tot het ene foute programma na het andere.

In dit opzicht vormt de nu volgende serie opgaven, gewild of niet, een ongebruikelijke oefening in wiskundige stijl.

(Einde Opmerking.)

(0) Schrijf een programma dat vaststelt of van drie getallen een der getallen het gemiddelde van de overige twee is.

(1) Schrijf een programma dat voor twee getallen  $p$  en  $q$ ,  $q \neq 0$ , een oplossing berekent van de vergelijking

$$x: \quad x \leq \frac{p}{q} < x + 1 \quad .$$

(2) Schrijf een programma ter berekening van het grootste achtvoud dat ten hoogste een gegeven getal is.

(3) Schrijf een programma ter bepaling van de cijfers van de zeventallige voorstelling van een natuurlijk getal kleiner dan 343 .

(4) Gegeven is een natuurlijk getal  $N$  ,  $N < 1024$  .

Bepaal het aantal nullen waarop de decimale voorstelling van  $N!$  eindigt.

```
(5)  S:  |[ N: int {N ≥ 0}
        , |[ r: int
          , S
          {r = (Σ i: 0 ≤ i < N: (-1)i · i)}
        ]|
      ]| .
```

```
(6)  S:  |[ N: int {N ≥ 0}; F(x: 0 ≤ x < N): array of int
        , |[ r: int
          , S
          {r = (Σ x: 0 ≤ x < N: (-1)x · F(x))}
        ]|
      ]| .
```

```
(7)  S:  |[ N: int {N ≥ 0}; F(x: 0 ≤ x < N): array of int
        , |[ r: int
          , S
          {r = (Σ x: 0 ≤ x ∧ x2 < N: F(x2))}
        ]|
      ]| .
```

(8) Schrijf een programma dat voor een gegeven getallenrij vaststelt of zij monotoon is.

(9) Bewijs de correctheid van

```

| [ a: int {a ≥ 0}
  ; | [ q, r: int
    ; q, r := 0, a + 1
    ; do r ≥ 4
      → | [ x, y: int
        ; x, y := r div 4, r mod 4
        ; q, r := q + x, x + y
      ] |
    od
    ; r := r - 1
    {a = 3 · q + r ∧ 0 ≤ r < 3}
  ] |
] | .

```

(10) S: | [ N, X: int {N ≥ 0}; H(n: 0 ≤ n < N): array of int  
 ; | [ r: int  
 ; S  
 {r = (S n: 0 ≤ n < N: H(n) · X<sup>n</sup>)}  
 ] |  
 ] | .

(11) Schrijf een programma dat, voor een gegeven getal N, de kleinste oplossing bepaalt van de vergelijking

$$k: \quad k \geq 0 \wedge 2^k \geq N .$$

(12) De rij  $F(x: x \geq 0)$  is gedefinieerd met het recurrentieschema

$$F(0) = 0, F(1) = 1,$$

$$F(x + 2) = F(x + 1) + F(x) \text{ voor alle } x \text{ met } x \geq 0.$$

Schrijf een programma dat voor gegeven  $N \geq 0$ ,  $F(N)$  berekent.

(13) De functie  $C(n: n \geq 0)$  is gedefinieerd met het recurrentieschema

$$C(0) = 0,$$

$$C(n) = C(n \text{ div } 10) + n \text{ mod } 10 \text{ voor elke } n \text{ met } n \geq 1.$$

Construeer  $S$  zo dat

```

| [ N: int {N ≥ 0}
  ; | [ c: int
    ; S
      {c = C(N)}
    ] |
  ] | .

```

Bewijs  $(\underline{A} N: N \geq 0: N \text{ mod } 9 = C(N) \text{ mod } 9)$ .

(14) Gegeven is een rij  $X(i: 0 \leq i < N)$ ,  $N \geq 0$ , zo dat  
 $(\underline{A} i: 0 \leq i < N: X(i) = 0 \vee X(i) = 1)$ .

Schrijf een programma ter berekening van de waarde van

$$(\underline{E} n: 0 \leq n \leq N: (\underline{A} i: 0 \leq i < n: X(i) = 0) \wedge (\underline{A} i: n \leq i < N: X(i) = 1)).$$

(15)  $S: | [ N: int \{N \geq 0\}$

    ;  $F(x: 0 \leq x < N): \underline{\text{array of int}}$

$\{(\underline{A} x: 0 \leq x < N: F(x) = 0 \vee F(x) = 1)\}$

    ;  $| [ c: int$

```

    ; S
    {c = (N x, y:  $0 \leq x < y < N$ :  $F(x) < F(y)$ )}
  ]|
]| .

```

```

(16)  S: |[ N: int { $N \geq 0$ }
      ; F(x:  $0 \leq x < N + 1$ ): array of int
      ; |[ b: bool
        ; S
        {b  $\equiv$  (E x:  $0 \leq x < N$ :  $F(x) = F(N)$ )}
      ]|
    ]| .

```

(17) Schrijf een programma dat vaststelt of in een gegeven getallenrij het getal 7 voorkomt.

(18) Schrijf een programma dat het aantal verschillende delers van een positief getal bepaalt.

```

(19)  S: |[ r, k: int
      {r mod 2 = 1  $\wedge$   $1 \leq r < 2^k$ }
      ; |[ x: int
        ; S
        {x mod 2 = 1  $\wedge$   $1 \leq x < 2^k$   $\wedge$   $(x^3 - r) \text{mod } 2^k = 0$ }
      ]|
    ]| .

```

```

(20)  S:  |[ N: int {N ≥ 0}; X(i: 0 ≤ i < N): array of int
        ; |[ c: int
        ; S
        {c = (N i, j: 0 ≤ i < j < N: X(i) ≤ 0 ∧ X(j) ≥ 2)}
      ]|
    ]|

```

(21) Van drie ascending rijen  $F, G, H(i: i \geq 0)$  is gegeven dat  $(\exists i, j, k: i \geq 0 \wedge j \geq 0 \wedge k \geq 0: F(i) = G(j) = H(k))$ . Schrijf een programma ter berekening van het kleinste getal dat in elk van de drie rijen voorkomt.

(22) Gegeven zijn twee ascending rijen  $X(i: 0 \leq i < M + 1)$  en  $Y(i: 0 \leq i < N + 1)$ ,  $M \geq 0$  en  $N \geq 0$ . Er geldt  $X(M) = Y(N)$ . Op een getallenrechte liggen  $M$  rode stipjes, genummerd van en met 0 tot en zonder  $M$ , en  $N$  blauwe stipjes, genummerd van en met 0 tot en zonder  $N$ .

De positie van rood stipje  $i$  is gegeven door  $X(i)$  en de positie van blauw stipje  $j$  door  $Y(j)$ .

Schrijf een programma dat vaststelt of er een rood en een blauw stipje zijn op een afstand van minder dan 7.

(23) Schrijf een programma dat voor een ascending rij het aantal verschillende in de rij voorkomende waarden bepaalt.

(24) Grigri:

```

|[ N: int {N ≥ 1}
; X(i: 0 ≤ i < N): array of int {X is ascending}

```

```

    , l[ lp: int
      ; Grigri
      {lp = (MAX i, j: 0 ≤ i ≤ j < N ∧ X(i) = X(j): j - i + 1)}
    ]l
  ]l .

```

(25) Gegeven is een rij  $X(i: 0 \leq i < N)$ ,  $N \geq 1$ .

Voor  $0 \leq i < j \leq N$  is  $H(i, j)$  gedefinieerd door

$$H(i, j) \equiv (\underline{A} k: i \leq k < j: X(i) = X(k)) .$$

Gevraagd wordt een programma ter berekening van

$$(\underline{N} i, j: 0 \leq i < j \leq N: H(i, j)) .$$

(26) Schrijf een programma dat voor een getal  $N \geq 1$  de periode in de decimale representatie van  $1 / N$  bepaalt.

(27) Met behulp van een natuurlijke functie  $f(x: x \geq 0)$  is de rij  $RF(n: n \geq 0)$  gedefinieerd:

$$RF(0) = 0, \quad RF(n + 1) = f(RF(n)) \quad \text{voor alle } n \geq 0 .$$

Gegeven is dat de rij  $RF$  op den duur periodiek is, i.e.

$$(\underline{E} i, j: 0 \leq i < j: RF(i) = RF(j)) .$$

Schrijf een programma ter berekening van de periode van de rij  $RF$ . De restrictie is dat daarbij geen gebruik mag worden gemaakt van een lokaal array.

(28) Schrijf een programma dat een tabel van de eerste  $N$  derde machten van natuurlijke getallen aanlegt. De enige daarbij toegestane aritmetische bewerkingen zijn optelling en aftrekking.

(29) De geheeltallige functie  $DG(i, j: 0 \leq i < I \wedge 0 \leq j < J)$  is zo dat voor elke  $i: 0 \leq i < I$  de rij  $DG(i, j: 0 \leq j < J)$  ascending is en zo dat voor elke  $j: 0 \leq j < J$  de rij  $DG(i, j: 0 \leq i < I)$  descending is.

Voor een gegeven getal  $X$  geldt

$$(\exists i, j: 0 \leq i < I \wedge 0 \leq j < J: X = DG(i, j)) .$$

Schrijf een programma dat een paar  $(r, s)$  berekent waarvoor  $0 \leq r < I \wedge 0 \leq s < J \wedge X = DG(r, s)$  .

(30) Schrijf, bij gegeven geheeltallige functie  $M(i, j: 0 \leq i < I \wedge 0 \leq j < J)$  , een programma ter berekening van

$$(\forall i, j: 0 \leq i < I \wedge 0 \leq j < J: M(i, j) \geq 0) ,$$

in elk der onderstaande gevallen:

- (a) Van  $M$  is niets naders gegeven
- (b)  $M(i, j)$  is decreasing als functie van  $i$  en increasing als functie van  $j$
- (c)  $M(i, j)$  is descending als functie van  $i$  en ascending als functie van  $j$  .

Hoe veranderen de programma's in het geval gevraagd wordt de waarde van

$$(\forall i, j: 0 \leq i < I \wedge 0 \leq j < J: M(i, j) = 0)$$

te berekenen.

(31) S:  $\llbracket M, N: \text{int} \{M \geq 0 \wedge N \geq 0\}$   
            $; F(i: 0 \leq i < M), G(j: 0 \leq j < N): \text{array of int}$   
            $\{(F \text{ is ascending}) \wedge (G \text{ is ascending})\}$   
            $; \llbracket c: \text{int}$

```

; S
{c = (N i, j:  $0 \leq i < M \wedge 0 \leq j < N$ :
      F(i) + G(j) > 0)}
]|
]| .

```

```

(32) S: |[ N: int {N ≥ 0}
      ; U(i:  $0 \leq i < N$ ): array of int
        {[A i:  $0 \leq i < N$ : U(i) > 0]}
      ; |[ r: int
        ; S
          {r = (N j, k:  $0 \leq j \leq N \wedge 0 \leq k \leq N$ :
                (S i:  $0 \leq i < j$ : U(i)) =
                (S i:  $k \leq i < N$ : U(i)))}
        ]|
      ]| .

```

(33) Met behulp van twee gehele getallen  $X$  en  $Y$  is de rij  $F(i: i \geq 0)$  gedefinieerd:

$$F(0) = 1, F(1) = 1,$$

$$F(i + 2) = X \cdot F(i) + Y \cdot F(i + 1) \text{ voor alle } i \geq 0.$$

Gevraagd wordt een oplossing voor

Fibolucci:

```

|[ N: int {N ≥ 1}
; |[ r: int
  ; Fibolucci
    {r = (S i:  $0 \leq i \leq N$ : F(i) • F(N - i))}
  ]|
]| .

```

(34) Schrijf een programma dat vaststelt of er een getal bestaat dat in elke van twee gegeven ascending getallenrijen voorkomt (the coincidence-test).

(35) Een segment  $X(i: p \leq i < q)$  van  $X(i: 0 \leq i < N)$  is "linksminimaal" betekent

$$0 \leq p < q \leq N \wedge (\forall i: p \leq i < q: X(p) \leq X(i)) .$$

Schrijf een programma ter berekening van de maximale lengte van enig linksminimaal segment van een gegeven rij  $X(i: 0 \leq i < N)$ ,  $N \geq 1$ .

(36) Een segment  $X(i: p \leq i < q)$  van  $X(i: 0 \leq i < N)$  is "bijna-ascending" betekent

$$0 \leq p < q \leq N \wedge (\exists i: p < i < q: X(i-1) > X(i)) \leq 1 .$$

Schrijf een programma ter berekening van de maximale lengte van enig bijna-ascending segment van een gegeven rij  $X(i: 0 \leq i < N)$ ,  $N \geq 1$ .

(37) Een segment  $X(i: p \leq i < q)$  van  $X(i: 0 \leq i < N)$  is een "K-segment" betekent

$$0 \leq p < q \leq N \wedge (\forall i, j: p \leq i < q \wedge p \leq j < q: X(i) \cdot X(j) \geq 0) .$$

Schrijf een programma ter berekening van de maximale lengte van enig K-segment van een gegeven rij  $X(i: 0 \leq i < N)$ ,  $N \geq 1$ .

(38) Een segment  $X(i: p \leq i < q)$  van  $X(i: 0 \leq i < N)$  is een "glad segment" betekent

$$0 \leq p < q \leq N \wedge (\forall i, j: p \leq i < q \wedge p \leq j < q: X(i) - X(j) \leq 1) .$$

Schrijf een programma ter berekening van de maximale lengte van enig glad segment van een gegeven rij  $X(i: 0 \leq i < N)$ ,  $N \geq 1$ .

(39) Een segment  $X(i: p \leq i < q)$  van  $X(i: 0 \leq i < N)$  is een "M-segment" betekent

$$0 \leq p < q \leq N \wedge (\underline{N} z: z \text{ geheel}: (\underline{E} i: p \leq i < q: z = X(i))) \leq 47.$$

Schrijf een programma ter berekening van de maximale lengte van enig M-segment van een gegeven rij  $X(i: 0 \leq i < N)$ ,  $N \geq 1$ .

(40) Een segment  $X(i: p \leq i < q)$  van  $X(i: 0 \leq i < N)$  is een "dip-segment" betekent

$$0 \leq p < q \leq N \wedge (\underline{A} i, j: p \leq i \leq j < q: X(i) \leq X(j) + 1).$$

Schrijf een programma ter berekening van de maximale lengte van enig dip-segment van een gegeven rij  $X(i: 0 \leq i < N)$ ,  $N \geq 1$ .

(41) Schrijf een programma dat voor gegeven  $N \geq 0$  de waarde berekent van  $(\underline{N} x, y: 0 \leq x \leq y: x^2 + y^2 = N)$ .

```
(42) S: |[ N, U: int {N ≥ 0 ∧ U ≥ 0}
      ; X(i: 0 ≤ i < N): array of int
      { X is ascending }
      ; |[ r: int
      ; S
      {r = (N i, j: 0 ≤ i ≤ j < N: X(j) - X(i) ≤ U)}
      ]|
    ]| .
```

(43) De Boolese functie  $B(x)$  is gedefinieerd voor alle gehele  $x$  en heeft in ten minste één punt de waarde `true`. Schrijf een programma dat zo'n punt bepaalt. (De expressie  $B(x)$  mag als Boolese expressie worden beschouwd.)

(44) Schrijf een programma  $S$  zo dat voldaan is aan

```

[[ N: int {N ≥ 0 ∧ N = NO}
  ; X(i: 0 ≤ i < N): array of int
    {(A i: 0 ≤ i < N: X(i) = 0 ∨ X(i) = 1)}
  ; S
    {(X is ascending) ∧ N = NO}
]] .

```

De restrictie hierbij is dat de enige toegelaten operaties op het array  $X$  zijn  $X: \text{swap}(i, j)$  met  $0 \leq i < N$  en  $0 \leq j < N$ .

(45) Gegeven is een functie  $f(x: 0 \leq x < N)$  waarvoor geldt  $(A x: 0 \leq x < N: 0 \leq f(x) < N)$ .

In een land zijn  $N$  plaatsen, genummerd van  $0$  tot en met  $N - 1$ . Zodra de gong gaat begeeft een persoon die zich in plaats  $x$  bevindt zich instantaan naar plaats  $f(x)$ .

Er zijn twee personen,  $C$  en  $D$ .  $C$ , die een brandende toorts draagt, bevindt zich aanvankelijk in plaats  $c$ , en  $D$ , met buskruit, in plaats  $d$ .

Dan volgt een onbeperkt aantal gongslagen.

Schrijf een programma dat aan de boolean variabele `bang` de passende waarde toekent.

Schrijf ook een programma voor het geval dat  $C$  loopt onder controle van  $f$  en  $D$  onder controle van een functie  $g$  met  $(A x: 0 \leq x < N: 0 \leq g(x) < N)$ .

(46) Een integer functie  $K(i, j: i \geq 0 \wedge j \geq 0)$  is increasing in beide argumenten.

Schrijf een programma ter berekening van

$(N \ i, j: i \geq 0 \wedge j \geq 0: 0 \leq K(i, j) < 47)$  .

(47) De syntactische categorie  $h$  is gedefinieerd door  
 $\langle h \rangle ::= 0 \mid 1 \langle h \rangle \langle h \rangle$  .

Construeer een programma  $S$  zo dat

```

| [ N: int {N ≥ 0}
  ; X(i: 0 ≤ i < 2 · N + 1): array of int
    { (A i: 0 ≤ i < 2 · N + 1: X(i) = 0 ∨ X(i) = 1) }
  ; | [ b: bool
    ; S
    { b ≡ (X behoort tot de categorie h) }
  ] |
] | .

```

Geef ook een  $S$  voor het geval  $h$  gegeven is door  
 $\langle h \rangle ::= 0 \mid \langle h \rangle 1 \langle h \rangle$  .

(48) De rij  $\text{fusc}(n: n \geq 0)$  is gedefinieerd door

$\text{fusc}(0) = 0$  ,  $\text{fusc}(1) = 1$  en voor alle  $n \geq 0$

$\text{fusc}(2 \cdot n) = \text{fusc}(n)$  en  $\text{fusc}(2 \cdot n + 1) = \text{fusc}(n) + \text{fusc}(n + 1)$  .

Bepaal een  $S$  zo dat

```

| [ N: int {N ≥ 0}
  ; | [ x: int; S {x = fusc(N)} ] |
] | .

```

(49) De verzameling  $W$  is gedefinieerd door

(a) 0 behoort tot  $W$

(b) als  $x$  tot  $W$  behoort dan behoren ook  $2 \cdot x + 1$  en  $3 \cdot x + 1$  tot  $W$

(c) alle elementen van  $W$  behoren tot  $W$  op grond van (a) of (b).

Schrijf een programma ter berekening van de kleinste  $N$  elementen van  $W$ , voor gegeven  $N \geq 1$ .

(50) De superpositie van een collectie intervallen op een getallenrechte is dat deel van de getallenrechte dat door deze intervallen bedekt wordt.

De lengte van zo een superpositie is de lengte van al hetgeen zwart ziet nadat van een aanvankelijk witte getallenrechte de superpositie zwart geschilderd is.

Schrijf een programmasegment  $S$  zo dat

```

| [ N: int {N ≥ 0}
  ; X, Y(i: 0 ≤ i < N): array of int
    {(X is ascending) ∧ (∃ i: 0 ≤ i < N: X(i) ≤ Y(i))}
  ; | l: int
    ; S
      {l = de lengte van de superpositie van de intervallen
        (X(i), Y(i)) , 0 ≤ i < N}
    ] |
] | .

```

(51) Gegeven is een positieve integer  $A$  en een rij  $f(i: i \geq 0)$  van natuurlijke getallen.

Er geldt

$(\underline{E} n: n \geq 0: (\underline{A} i: 0 \leq i < n: f(i) \leq A) \wedge (\underline{A} i: n \leq i: f(i) > A))$  .

Gevraagd wordt een programma ter berekening van

$(\underline{N} i, j: 0 \leq i < j: A = (\underline{S} k: i \leq k < j: f(k)))$  .

Schrijf ook een programma voor het speciale geval dat  $f(i) = i$  voor alle  $i \geq 0$  .

(52) De  $N$  hoekpunten van een veelhoek zijn (rechts) omgaand genummerd van en met 0 tot  $N$  . De afstand van hoekpunt  $i$  tot zijn omgaande buur is  $d(i)$  ,  $d(i) > 0$  .

Schrijf een programma ter bepaling van een hoekpuntenpaar dat de omtrek zo goed mogelijk halveert.

```
(53)  S:  |[ M, N: int {M ≥ 0 ∧ N ≥ 0}
        ; K(i: 0 ≤ i < M): array of int
        ; |[ b: bool
          ; S
            {b ≡ (A n: 0 ≤ n < N:
                  (E m: 0 ≤ m < M: K(m) = n))}
        ]|
      ]| .
```

(54) De  $N$  punten van een gerichte graaf zijn genummerd van en met 0 tot  $N$  ,  $N \geq 1$  .

De takken zijn gegeven door twee arrays  $p, q(i: 0 \leq i < N)$  : voor alle  $(i, j)$  ,  $0 \leq i < N \wedge 0 \leq j < N$  , geldt

er is een tak van  $i$  naar  $j$   
 $\equiv (i < j \wedge (p(i) = j \vee q(i) = j))$  .

Schrijf een programma ter berekening van het aantal paden (in de graaf) van punt 0 naar punt  $N - 1$  .

(55) Van een rij  $X(i: 0 \leq i < N)$ ,  $N \geq 0$ , is gegeven dat  
 $(\bigwedge i: 0 \leq i < N: X(i) = 0 \vee X(i) = 1)$ .

Een segment  $X(i: p \leq i < q)$  van  $X(i: 0 \leq i < N)$  is "gebalanceerd" betekent

$$0 \leq p \leq q \leq N \wedge (\sum_{i: p \leq i < q: X(i) = 0} = \sum_{i: p \leq i < q: X(i) = 1})$$

Schrijf een programma ter berekening van de maximale lengte van enig gebalanceerd segment van  $X(i: 0 \leq i < N)$ .

(56) Gegeven zijn twee rijen positieve getallen  $p, q(i: 0 \leq i < N)$ , waarvoor geldt

$$(\sum_{i: 0 \leq i < N: p(i)}) = (\sum_{i: 0 \leq i < N: q(i)})$$

Langs een cirkelvormige race-baan liggen  $N$  pits, (rechts)omgaand genummerd van en met 0 tot  $N$ . In pit  $i$  is een hoeveelheid benzine, groot  $p(i)$ , aanwezig. De hoeveelheid benzine nodig om een race-auto de afstand van pit  $i$  naar de omgaande buurpit te laten overbruggen is gelijk aan  $q(i)$ .

Schrijf een programma dat berekent vanuit hoeveel pits een race-auto met een aanvankelijk lege maar voldoende grote tank de race-baan rond kan.

(57) Gegeven zijn twee rijen  $X, Y(i: 0 \leq i < N)$ .

Geef een programma dat de lexicografische volgorde van  $X$  en  $Y$  bepaalt, i.e. vaststelt of  $X < Y$ , dan wel  $X = Y$ , dan wel  $X > Y$ .

(58) De ascending rijen van positieve getallen en met som  $N$  kunnen lexicografisch gerangschikt worden.

Schrijf een programma dat zo'n rij, die niet de lexicografisch laatste is, transformeert in zijn lexicografische successor.

(59) Gegeven is een natuurlijke functie  $f(x, y: x \geq 0 \wedge y \geq 0)$  die voldoet aan

(i)  $f(x, y) > x$  voor alle  $x$  en  $y$

(ii)  $y_1 > y_0 \Rightarrow f(x, y_1) > f(x, y_0)$  voor alle  $x, y_0$  en  $y_1$ .

Schrijf een programma dat voor een gegeven  $N \geq 1$  de kleinste  $N$  elementen berekent van de verzameling  $V$  gedefinieerd door

(a) 0 behoort tot  $V$

(b) als  $x$  en  $y$  tot  $V$  behoren, dan behoort ook  $f(x, y)$  tot  $V$

(c) alle elementen die tot  $V$  behoren, behoren tot  $V$  op grond van (a) of (b).

(60) De positieve functie  $C(x, y: x \geq 1 \wedge y \geq 0)$  is gedefinieerd met het recurrentieschema

$C(1, y) = 1$  voor alle  $y \geq 0$  ;

$C(x + 1, y) = C(x, y)$  voor alle  $x, y$  met  
 $x \geq 1$  en  $0 \leq y \leq x$  ;

$C(x + 1, y) = C(x, y) + C(x + 1, y - (x + 1))$   
voor alle  $x, y$  met  $x < y$  en  $x \geq 1$  .

Schrijf een programma dat voor een gegeven  $N \geq 1$  de waarde van  $C(N, N)$  berekent.

(61) Een (niet-lege, eindige) boom is gegeven door

(a) punt  $r$  is de wortel van de boom

- (b) een punt  $x$  van de boom heeft  $n(x)$  zonen, te weten de punten  $s(x, i)$  met  $0 \leq i < n(x)$ .

Een blad is een punt van de boom zonder zonen.

Schrijf een programma dat het aantal bladeren van de boom bepaalt (tip-count).

- (62) Gegeven zijn een integer  $K$ ,  $K \geq 0$ , en een integer array  $X(i: 0 \leq i < N)$ .

Gevraagd wordt een programma ter berekening van de maximale lengte van enige aaneengesloten deelrij van  $X$  die ten hoogste  $K$  nullen bevat.

De restrictie hierbij is dat de elementen van  $X$  slechts één keer geïnspecteerd mogen worden.

- (63) Gegeven is een matrix waarvan alle elementen 0 of 1 zijn. Gevraagd wordt een programma ter berekening van de maximale omvang van enige vierkante deelmatrix waarvan alle elementen gelijk aan 0 zijn.

- (64) Gegeven zijn twee integer arrays  $X, Y(i: 0 \leq i < 47)$ .

In het platte vlak liggen 47 punten, genummerd van en met 0 tot 47. Punt  $i$  heeft  $(X(i), Y(i))$  als Cartesische coördinaten.

Een robot maakt een rondgang langs de punten in volgorde van opklimmend nummer en keert vanuit punt 46 weer terug naar punt 0. Daarbij neemt hij de volgende regels in acht:

- (a) hij start in punt 0 kijkend in de richting van punt 1
- (b) hij loopt steeds in de richting waarin hij kijkt

(c) hij verandert zijn richting slechts in de punten 1 ,  
namelijk door een rechtsomgaande rotatie over een hoek  
kleiner dan  $360^\circ$

(d) hij eindigt in punt 0 kijkend in de richting van punt 1 .

Ten gevolge van deze rondgang maakt de robot een geheel aantal  
volledige (rechtsomgaande) omwentelingen.

Schrijf een programma ter berekening van dit aantal, met als  
restrictie dat elke expressie in het programma van het type integer  
of boolean dient te zijn.

(Einde 2 Programmeeropgaven.)

### 3 ENKELE UITWERKINGEN

Ter illustratie van hoe oplossingen er uit zouden kunnen zien geven we een aantal voorbeelden.

#### 0.0 Opgave 3

$$\begin{aligned}
 & (\neg P \vee Q) \wedge (P \vee R) \\
 = & \quad \{\text{distributie van conjunctie over disjunctie}\} \\
 & ((\neg P \vee Q) \wedge P) \vee ((\neg P \vee Q) \wedge R) \\
 = & \quad \{\text{complementregels}\} \\
 & (Q \wedge P) \vee ((\neg P \vee (P \wedge Q)) \wedge R) \\
 = & \quad \{\text{distributie van conjunctie over disjunctie}\} \\
 & (Q \wedge P) \vee (\neg P \wedge R) \vee (P \wedge Q \wedge R) \\
 = & \quad \{\text{absorptieregel}\} \\
 & (Q \wedge P) \vee (\neg P \wedge R) \\
 \text{(Einde 0.0 Opgave 3.)}
 \end{aligned}$$

#### 0.0 Opgave 16

De domeinen voor de dummies  $i$  en  $j$  zijn voor de duur van het betoog constant. We laten ze anoniem.

$$\begin{aligned}
 & \underline{A} \ i :: (\underline{A} \ j :: X(i) \cdot X(j) \geq 0) \\
 = & \quad \{\text{rekenkunde}\} \\
 & \underline{A} \ i :: (\underline{A} \ j :: (X(i) \geq 0 \vee X(j) \leq 0) \wedge (X(i) \leq 0 \vee X(j) \geq 0))
 \end{aligned}$$

$$\begin{aligned}
&= \{(\underline{A} \ j:: P \wedge Q) \equiv (\underline{A} \ j:: P) \wedge (\underline{A} \ j:: Q)\} \\
&\quad (\underline{A} \ i:: (\underline{A} \ j:: X(i) \geq 0 \vee X(j) \leq 0) \\
&\quad \quad \wedge (\underline{A} \ j:: X(i) \leq 0 \vee X(j) \geq 0) \\
&\quad ) \\
&= \{(\underline{A} \ i:: P \wedge Q) \equiv (\underline{A} \ i:: P) \wedge (\underline{A} \ i:: Q)\} \\
&\quad (\underline{A} \ i:: (\underline{A} \ j:: X(i) \geq 0 \vee X(j) \leq 0)) \\
&\quad \wedge (\underline{A} \ i:: (\underline{A} \ j:: X(i) \leq 0 \vee X(j) \geq 0)) \\
&= \{\text{symmetrie}\} \\
&\quad (\underline{A} \ i:: (\underline{A} \ j:: X(i) \geq 0 \vee X(j) \leq 0)) \\
&= \{j \text{ komt niet voor in } X(i) \geq 0\} \\
&\quad (\underline{A} \ i:: X(i) \geq 0 \vee (\underline{A} \ j:: X(j) \leq 0)) \\
&= \{i \text{ komt niet voor in } (\underline{A} \ j:: X(j) \leq 0)\} \\
&\quad (\underline{A} \ i:: X(i) \geq 0) \vee (\underline{A} \ j:: X(j) \leq 0) \quad .
\end{aligned}$$

(Einde 0.0 Opgave 16.)

### 0.3 Opgave 1(c)

Op grond van

$$\begin{aligned}
&X \vee (P \wedge Q) \Rightarrow Q \\
&= \{\text{definitie } \Rightarrow\} \\
&\quad \neg(X \vee (P \wedge Q)) \vee Q \\
&= \{\text{de Morgan}\} \\
&\quad (\neg X \wedge (\neg P \vee \neg Q)) \vee Q \\
&= \{\text{distributie van disjunctie over conjunctie}\} \\
&\quad (\neg X \vee Q) \wedge (\neg P \vee \neg Q \vee Q) \\
&= \{\text{elementair}\} \\
&\quad \neg X \vee Q \\
&= \{\text{definitie } \Rightarrow\} \\
&\quad X \Rightarrow Q
\end{aligned}$$

kan de gegeven vergelijking herschreven worden tot

$$X: X \Rightarrow Q$$

met  $Q$  als zwakste oplossing.

(Einde 0.3 Opgave 1(c).)

### 0.3 Opgave 6

Voor elk tweetal rijen  $f, g(i: 0 \leq i < N)$  geldt

$$\begin{aligned}
 & \neg(f < g) \wedge \neg(g < f) \\
 = & \quad \{\text{definitie lexicografische ordening; de Morgan}\} \\
 & (\underline{A} x: 0 \leq x < N: f(x) \geq g(x) \vee (\underline{E} y: 0 \leq y < x: f(y) \neq g(y))) \\
 & \wedge \\
 & (\underline{A} x: 0 \leq x < N: g(x) \geq f(x) \vee (\underline{E} y: 0 \leq y < x: g(y) \neq f(y))) \\
 = & \quad \{(\underline{A} x: P: Q) \wedge (\underline{A} x: P: R) \equiv (\underline{A} x: P: Q \wedge R)\} \\
 & (\underline{A} x: 0 \leq x < N: (f(x) \geq g(x) \vee (\underline{E} y: 0 \leq y < x: f(y) \neq g(y))) \\
 & \quad \wedge (g(x) \geq f(x) \vee (\underline{E} y: 0 \leq y < x: g(y) \neq f(y))) \\
 & ) \\
 = & \quad \{\text{distributie disjunctie over conjunctie; aritmetiek}\} \\
 & (\underline{A} x: 0 \leq x < N: f(x) = g(x) \vee (\underline{E} y: 0 \leq y < x: f(y) \neq g(y))) \\
 = & \quad \{\text{volledige inductie}\} \\
 & (\underline{A} x: 0 \leq x < N: f(x) = g(x)) \\
 = & \quad \{\text{definitie gelijkheid van rijen}\} \\
 & (f = g) \quad .
 \end{aligned}$$

(Einde 0.3 Opgave 6.)

### 1 Opgave 4

Opdat geldt

$$| [x, y, m: \text{int } \{x \geq 0 \wedge y \geq 0\}; m := x + y \{m \geq x \max y\}] | ,$$

dient voldaan te zijn aan

$$x \geq 0 \wedge y \geq 0 \Rightarrow (m \geq x \max_{x+y} y)_{x+y}^m ,$$

en dit is zo wegens

$$\begin{aligned}
 & (m \geq x \max y)_{x+y}^m \\
 = & \{ \text{definitie } R_E^x \} \\
 & x + y \geq x \max y \\
 & \{ \text{definitie } x \max y \} \\
 & x + y \geq x \wedge x + y \geq y \\
 = & \{ \text{aritmetiek} \} \\
 & x \geq 0 \wedge y \geq 0 .
 \end{aligned}$$

(Einde 1 Opgave 4.)

## 1 Opgave 8

De geldigheid van

$$| [ x, y : \text{int } \{P\}; x := (x - y) / 2; x := 2 * x + y \{P\} ] |$$

betekent, op grond van de regels van de concatenatie en de assignment, dat  $P$  voldoet aan

$$P \Rightarrow (x - y \text{ even}) \wedge (P_{2 \cdot x + y}^x)_{(x-y)/2}^x ,$$

ofwel, wegens  $(P_{2 \cdot x + y}^x)_{(x-y)/2}^x \equiv P$ , aan

$$P \Rightarrow (x - y \text{ even}) \wedge P$$

ofwel aan

$$P \Rightarrow (x - y \text{ even}) .$$

(Einde 1 Opgave 8.)

## 1 Opgave 26

De geldigheid van

```

| [ x, y: int
  {P}; if B0 → x, y := y, x  []  B1 → x := x - y fi {P}
] |

```

betekent, op grond van de regels van de alternatieve statement en de assignment, dat voldaan is aan elk der voorwaarden

$$P \Rightarrow B0 \vee B1$$

$$(i) \quad P \wedge B0 \Rightarrow P_{y,x}^{x,y}$$

$$(ii) \quad P \wedge B1 \Rightarrow P_{x-y}^x .$$

Met  $P$  gedefinieerd door  $P \equiv x > 0 \wedge y > 0$  betekent dit voor de Boolese expressies  $B0$  en  $B1$  dat zij dienen te voldoen aan elk der voorwaarden

$$x > 0 \wedge y > 0 \Rightarrow B0 \vee B1$$

$$x > 0 \wedge y > 0 \wedge B0 \Rightarrow y > 0 \wedge x > 0$$

$$x > 0 \wedge y > 0 \wedge B1 \Rightarrow x - y > 0 \wedge y > 0 .$$

Hieraan is onder andere voldaan door de keuze  $true$  voor  $B0$  en  $x > y$  voor  $B1$  .

\*   \*   \*

De geldigheid van

```

| [ x, y: int
  {P ∧ x + 2 · y = D}
; if C0 → x, y := y, x  []  C1 → x := x - y fi
  {x + 2 · y < D}
] |

```

betekent dat voldaan is aan elk der voorwaarden

$$P \wedge x + 2 \cdot y = D \Rightarrow C0 \vee C1$$

$$(iii) \quad P \wedge x + 2 \cdot y = D \wedge C0 \Rightarrow (x + 2 \cdot y < D)_{y,x}^{x,y}$$

$$(iv) \quad P \wedge x + 2 \cdot y = D \wedge C1 \Rightarrow (x + 2 \cdot y < D)_{x-y}^x$$

Met  $P$  weer gedefinieerd door  $P \equiv x > 0 \wedge y > 0$  is hieraan voldaan door  $x < y$  voor  $C0$  en  $true$  voor  $C1$ .

\*   \*   \*

De conjunctie van (i) met (iii) en van (ii) met (iv) nemend, levert

$$P \wedge x + 2 \cdot y = D \wedge B0 \wedge C0 \Rightarrow (P \wedge x + 2 \cdot y < D)_{y,x}^{x,y}$$

$$P \wedge x + 2 \cdot y = D \wedge B1 \wedge C1 \Rightarrow (P \wedge x + 2 \cdot y < D)_{x-y}^x$$

ofwel, met de gemaakte keuzen voor  $B0$ ,  $B1$ ,  $C0$  en  $C1$ ,

$$P \wedge x + 2 \cdot y = D \wedge x < y \Rightarrow (P \wedge x + 2 \cdot y < D)_{y,x}^{x,y}$$

$$\text{en } P \wedge x + 2 \cdot y = D \wedge x > y \Rightarrow (P \wedge x + 2 \cdot y < D)_{x-y}^x,$$

waaruit samen met de geldigheid van

$$P \wedge (x < y \vee x > y) \Rightarrow x + 2 \cdot y \geq 0$$

uit het postulaat van de repetitieve statement (invariant  $P$ , variante functie  $x + 2 \cdot y$ ) volgt

```

| [ x, y: int
  {P}
  ; do x < y  $\rightarrow$  x, y := y, x  $\square$  x > y  $\rightarrow$  x := x - y od
  {P  $\wedge$  x = y}
]|

```

(Einde 1 Opgave 26.)

## 2 Opgave 13

Aan de gegeven functionele specificatie is voldaan met voor S het binnenblok

```

| [ n: int {N ≥ 0}
  ; c := 0; n := N
    {invariant P: 0 ≤ n ≤ N ∧ c + C(n) = C(N), zie Noot0 .
      variante functie: n
    }
  ; do n ≠ 0
    → c := c + n mod 10; n := n div 10 {P, zie Noot2}
  od
    {c = C(N), zie Noot1}
| ] .

```

Noot0 De geldigheid van

$$\begin{aligned}
 N \geq 0 & \Rightarrow (P_N^n)_0^c \text{ volgt uit} \\
 & (P_N^n)_0^c \\
 = & \{ \text{definitie } P \} \\
 & 0 \leq N \leq N \wedge 0 + C(N) = C(N) \\
 = & \{ \text{calculus} \} \\
 & 0 \leq N
 \end{aligned}$$

(Einde Noot0.)

Noot1 De geldigheid van

$$\begin{aligned}
 P \wedge n = 0 & \Rightarrow c = C(N) \text{ , volgt uit} \\
 & P \wedge n = 0 \\
 \Rightarrow & \{ \text{definitie } P \} \\
 & c + C(0) = C(N)
 \end{aligned}$$

= {definitie C}

$c = C(N)$

{Einde Noot1.}

Noot2 De geldigheid van

$P \wedge n \neq 0 \Rightarrow (P_{n \text{ div } 10}^n)^c + n \text{ mod } 10$  volgt uit

$(P_{n \text{ div } 10}^n)^c + n \text{ mod } 10$   
 = {definitie P}

$0 \leq n \text{ div } 10 \leq N \wedge c + n \text{ mod } 10 + C(n \text{ div } 10) = C(N)$

← {rekenkunde en definitie C}

$c + C(n) = C(N) \wedge 1 \leq n \leq N$

← {definitie P}

$P \wedge n \neq 0$

{Einde Noot2.}

Dat de repetitie eindigt volgt uit de geldigheid van

•  $P \Rightarrow n \geq 0$

(die de begrensdsheid, naar beneden, van de variante functie uitdrukt) en uit de geldigheid van

$P \wedge n \neq 0 \wedge n = VF \Rightarrow ((n < VF)^n_{n \text{ div } 10})^c + n \text{ mod } 10$

(die uitdrukt dat de variante functie effectief zakt), volgend uit

$((n < VF)^n_{n \text{ div } 10})^c + n \text{ mod } 10$

= {substitutie}

$n \text{ div } 10 < VF$

← {definitie div}

$n > 0 \wedge n = VF$

← {definitie P}

$P \wedge n \neq 0 \wedge n = VF$

Twee bedenkingen

(a) De bewering dat een gegeven programmatekst aan een gegeven functionele specificatie voldoet dient te worden opgevat als een wiskundige stelling en vergt derhalve een bewijs.

In deze betekenis is de bovenstaande presentatie van een uiterst vertrouwde en traditionele vorm: eerst wordt het programma gegeven en dan volgt het correctheidsbewijs. Er zijn allerlei --hier niet te bespreken-- omstandigheden te bedenken waaronder zulk een stijl van programmadocumentatie heel praktisch is. Als het er evenwel om gaat programma's te ontwerpen, i.e. wiskundige stellingen te ontwerpen, dan is deze vorm van overdracht wel erg ambachtelijk, immers er wordt nauwelijks melding gemaakt van het 'quo modo'. In de uitwerking van volgende opgaven zullen wij daar allengs meer aandacht aan besteden.

(b) Bovenstaand correctheidsbewijs is gegeven in ongeveer het fijnste detail dat het door ons beschreven redeneerwezen toelaat. Dat maakt de discussie van zo'n eenvoudig programmaatje wel wat erg lang. Wij hebben het bewijs in deze volle omvang dan ook alleen maar vertoond bij wijze van illustratie.

Naarmate we geoefender raken zullen we de fijnheid van onze bewijsstappen allengskens homogeen vergroven, daarbij nimmer uit het oog verliezend dat wij, indien daartoe uitgedaagd, een bewijs steeds in al zijn relevante detail moeten kunnen leveren.

(Einde Twee bedenkingen.)

De lezer verifiëre dat de relatie

$$(c + n) \bmod 9 = N \bmod 9$$

een invariant is voor de repetitie van het gegeven programma. Daaruit volgt dan samen met  $P$  en  $n = 0$  dat  $C(N) \bmod 9 = N \bmod 9$ .

(Einde 2 Opgave 13.)

## 2 Opgave 25

Een functionele specificatie van het te construeren programma is

```
S: |[ N: int {N ≥ 1}; X(i: 0 ≤ i < N): array of int
    ; |[ c: int
      ; S
      {R: c = (N i, j: 0 ≤ i < j ≤ N: H(i, j))}
    ]|
  ]| .
```

Hierin is  $R$  de naam voor de gewenste eindconditie en is  $H(i, j)$  een predicaat dat voor alle  $i, j$  met  $0 \leq i < j \leq N$  gedefinieerd is door  $H(i, j) \equiv (\underline{A} k: i \leq k < j: X(i) = X(k))$ .

Voor  $S$  kiezen we een binnenblok met repetitie. Als invariant van de repetitie kiezen we  $P$ , gedefinieerd door

$$P \equiv 1 \leq n \leq N \wedge c = (\underline{N} i, j: 0 \leq i < j \leq n: H(i, j)) ,$$

en verkregen uit  $R$  door in  $R$  de constante  $N$  te vervangen door de variabele  $n$ .

Geprojecteerd op  $n$  ziet  $S$  er uit als

$$|[ n: int; n := 1; \underline{do} n \neq N \rightarrow n := n + 1 \underline{od} ]| ,$$

zodat beëindiging gegarandeerd is (variante functie  $N - n$ ).

Omdat, gebruik makend van  $P \wedge n \neq N$

$$\begin{aligned} & (\underline{N} i, j: 0 \leq i < j \leq n + 1: H(i, j)) \\ = & \quad \{ \text{afsplitsen van de term met } j = n + 1, \text{ hetgeen geoorloofd} \\ & \quad \text{is wegens } 1 \leq n + 1 \leq N, \text{ volgend uit } P \wedge n \neq N \} \\ & (\underline{N} i, j: 0 \leq i < j \leq n: H(i, j)) \\ & + (\underline{N} i: 0 \leq i < n + 1: H(i, n + 1)) \\ = & \quad \{ \text{met } P \} \\ & c + (\underline{N} i: 0 \leq i < n + 1: H(i, n + 1)) , \end{aligned}$$

ziet  $S$ , geprojecteerd op  $n$  en  $c$ , er uit als

```

[[ n: int
  ; n:= 1; c:= 1 {P}
  ; do  $n \neq N \rightarrow c:= c + d; n:= n + 1$  od
]] ,

```

mits de preconditionie van  $c:= c + d$  versterkt kan worden met  $d = (\underline{N} \ i: 0 \leq i < n + 1: H(i, n + 1))$ .

Deze versterking bewerkstelligen we door de invariant te versterken tot  $P \wedge Q$ , waarin  $Q$  gedefinieerd is door

$$Q \equiv d = (\underline{N} \ i: 0 \leq i < n: H(i, n)) .$$

Omdat, gebruik makend van  $P \wedge Q \wedge n \neq N$ :

$$\begin{aligned}
 & (\underline{N} \ i: 0 \leq i < n + 1: H(i, n + 1)) \\
 = & \quad \{ \text{afsplitsen van de term met } i = n, \text{ hetgeen geoorloofd is} \\
 & \quad \text{wegens } 1 \leq n + 1 \leq N, \text{ volgend uit } P \wedge n \neq N \} \\
 & (\underline{N} \ i: 0 \leq i < n: H(i, n + 1)) + (\underline{N} \ i: 0 \leq i = n: H(n, n + 1)) \\
 = & \quad \{ \text{met de definitie van } H \} \\
 & (\underline{N} \ i: 0 \leq i < n: H(i, n) \wedge X(i) = X(n)) + \\
 & \quad (\underline{N} \ i: 0 \leq i = n: \text{true}) \\
 = & \quad \{ \text{Nootje beneden} + \text{calculus} \} \\
 & (\underline{N} \ i: 0 \leq i < n: H(i, n) \wedge X(n - 1) = X(n)) + 1 \\
 = & \quad \{ \text{calculus} \} \\
 & \underline{\text{if}} \ X(n - 1) = X(n) \rightarrow (\underline{N} \ i: 0 \leq i < n: H(i, n)) + 1 \\
 & \quad \underline{\text{fi}} \ X(n - 1) \neq X(n) \rightarrow (\underline{N} \ i: 0 \leq i < n: \text{false}) + 1 \\
 & \underline{\text{fi}} \\
 = & \quad \{ \text{met } Q \} \\
 & \underline{\text{if}} \ X(n - 1) = X(n) \rightarrow d + 1 \quad \underline{\text{fi}} \ X(n - 1) \neq X(n) \rightarrow 1 \underline{\text{fi}} ,
 \end{aligned}$$

ziet  $S$ , geprojecteerd op  $n$ ,  $c$  en  $d$ , er uit als

```

[[ n, d: int
  ; n:= 1; c:= 1 {P}; d:= 1 {P ∧ Q}
  ; do  $n \neq N$ 

```

```

→ {P ∧ Q ∧ n ≠ N}
  if X(n - 1) = X(n) → d := d + 1
  [] X(n - 1) ≠ X(n) → d := 1
  fi
  {P ∧ Qn+1n}
; c := c + d
  {Pn+1n ∧ Qn+1n}
; n := n + 1
  {P ∧ Q}
od
{P ∧ n = N, dus R}
]l .

```

Nootje Omdat voor  $0 \leq i < n$  geldt

$$H(i, n) \equiv H(i, n) \wedge X(i) = X(n - 1) , \quad (0)$$

concluderen we

$$\begin{aligned}
 & H(i, n) \wedge X(i) = X(n) \\
 = & \{ (0) \} \\
 & H(i, n) \wedge X(i) = X(n - 1) \wedge X(i) = X(n) \\
 = & \{ \text{rekenkunde} \} \\
 & H(i, n) \wedge X(i) = X(n - 1) \wedge X(n - 1) = X(n) \\
 = & \{ (0) \} \\
 & H(i, n) \wedge X(n - 1) = X(n) .
 \end{aligned}$$

(Einde Nootje.)

### Opmerkingen

a) De manier waarop P uit R verkregen is, is een heel vaak toepasbare --inmiddels standaard-- manier.

b) De manier waarop Q uit P verkregen is, is een heel vaak toepasbare --inmiddels standaard-- manier: Q registreert de wegens het behoud van P noodzakelijke verandering in c wanneer

n met 1 toeneemt. (In de wandelgangen heet  $Q$  wel eens de eerste afgeleide van  $P$ .)

c) De berekening die in het Nootje wordt uitgevoerd hadden we kunnen substitueren in de berekening die naar het Nootje verwijst. Dat zou dan wel aanleiding hebben gegeven tot veel copieerwerk omdat de formule die in het Nootje wordt gemasseerd slechts een klein onderdeel is van de formule waarin zij is ingebed.

d) Het stellinkje dat in het Nootje bewezen wordt is zo triviaal, dat het maar de vraag is of het tot in zulk detail bewezen dient te worden.

(Einde Opmerkingen.)

(Einde 2 Opgave 25.)

## 2 Opgave 30(c)

Met  $C$  gedefinieerd door

$$C = \{ \langle i, j \rangle : 0 \leq i < I \wedge 0 \leq j < J : M(i, j) \geq 0 \} ,$$

construeren wij een programma  $S$  zo dat voldaan is aan

```

| [ I, J: int {I ≥ 0 ∧ J ≥ 0}
  ; M(i, j: 0 ≤ i < I ∧ 0 ≤ j < J): array of int
    { (A i: 0 ≤ i < I: M(i, j: 0 ≤ j < J) is ascending)
      ∧ (A j: 0 ≤ j < J: M(i, j: 0 ≤ i < I) is descending) }
  ; | [ c: int
    ; S
      {R: c = C}
    ] |
  ] | .

```

Voor  $S$  kiezen wij een binnenblok met repetitie en voor de invariant de relatie  $P$  gedefinieerd door

$$\begin{aligned}
 P \equiv & (0 \leq m \leq I \wedge 0 \leq n \leq J \\
 & \wedge C = c + (\sum_{i, j: m \leq i < I \wedge n \leq j < J: M(i, j) \geq 0} 1) \\
 & ) .
 \end{aligned}$$

Aangezien de initialisatie van  $P$  gemakkelijk is voor  $m$  en  $n$  beide 0 en aangezien

$$P \wedge (m = I \vee n = J) \Rightarrow R$$

onderzoeken wij verhogingen van  $m$  en  $n$ .

Er geldt

$$\begin{aligned}
 & (\sum_{i, j: m+1 \leq i < I \wedge n \leq j < J: M(i, j) \geq 0}) \\
 = & \quad \{\text{calculus}\} \\
 & (\sum_{i, j: m \leq i < I \wedge n \leq j < J: M(i, j) \geq 0}) \\
 - & (\sum_{j: n \leq j < J: M(m, j) \geq 0}) .
 \end{aligned}$$

Van deze laatste expressie staat de eerste term in  $P$ , zodat we ons daar niet meer om hoeven te bekommeren. Voor de tweede term geldt, aannemende dat  $M(m, n) \geq 0$ ,

$$\begin{aligned}
 & (\sum_{j: n \leq j < J: M(m, j) \geq 0}) \\
 = & \quad \{\text{wegens de ascendingness van } M \text{ in het tweede argument} \\
 & \quad \text{geldt } M(m, n) \geq 0 \Rightarrow (\sum_{j: n \leq j < N: M(m, j) \geq 0})\} \\
 & J - n .
 \end{aligned}$$

Zodoende vinden we dat

$$\begin{aligned}
 & \{P \wedge m \neq I \wedge n \neq J \wedge M(m, n) \geq 0\} \\
 & c := c + J - n; m := m + 1 \\
 & \{P\} .
 \end{aligned}$$

Voor het geval  $M(m, n) < 0$  proberen we de term  $(\sum_{j: n \leq j < J: M(m, j) \geq 0})$  vooralsnog niet uit te rekenen, omdat dat veel te ingewikkeld is. Wij bekijken eerst een verhoging van  $n$ .

Op volkomen analoge wijze, maar nu gebruik makend van de descendingness van  $M$  in het eerste argument, vinden we

```

{P  $\wedge$  m  $\neq$  I  $\wedge$  n  $\neq$  J  $\wedge$  M(m, n) < 0}
n := n + 1
{P} ,

```

zodat we voor het binnenblok S mogen kiezen

```

| [ m, n: int
  ; c, m, n := 0, 0, 0
    {invariant P , variante functie (I - m) + (J - n)}
  ; do m  $\neq$  I  $\wedge$  n  $\neq$  J
     $\rightarrow$  if M(m, n)  $\geq$  0  $\rightarrow$  c := c + J - n; m := m + 1
      [] M(m, n) < 0  $\rightarrow$  n := n + 1
    fi
  od
] | .

```

(Einde 2 Opgave 30(c).)

(Einde 3 Enkele uitwerkingen.)

## ACADEMIC SERVICE INFORMATICA UITGAVEN

### INLEIDINGEN

*Computers en onze samenleving* van M.A. Arbib

*Basiskennis informatieverwerking* van Jan Everink

*De viewdata revolutie* van S. Fedida en R. Malik

*De informatiemaatschappij* van Jan Everink

*Informatica, een theoretische inleiding* van dr. L.P.J. Groenewegen en prof.dr. A. Ollongren

*AIV, Automatisering van de informatieverzorging* van ir. Th.J.G. Derksen en drs. H.W. Crins

*Organisatie, informatie en computers* van David M. Kroenke

### MICROCOMPUTERS

*Programmeercursus Microsoft BASIC* van Nok van Veen

*Werken met bestanden in BASIC* van L. Finkel en J.R. Brown

*Werken met bestanden op de Apple* van L. Finkel en J.R. Brown

*Werken met Visicalc* van C. Klitzner en M.J. Plociak, Jr.

*CP/M: een gids voor zelfstudie* van J.N. Fernandez en R. Ashley

*Cursus Z-80 assembleertaal* van Roger Hatty

*Exidy sorcerer en BASIC* van Nok van Veen e.a.

*TRS-80 BASIC: een gids voor zelfstudie* van Bob Albrecht e.a.

*TRS-80 BASIC voor gevorderden* van Don Inman e.a.

*Ontdek de ZX-Spectrum* van Tim Hartnell

*Flitsend FORTH* van Alan Winfield

### PROGRAMMEREN/PROGRAMMEERTALEN

*Een methode van programmeren* van prof.dr. Edsger W. Dijkstra en ir. W.H.J. Feijen

*Programmeren, het ontwerpen van algoritmen, in Pascal* van ir. J.J. van Amstel

*Inleiding tot het programmeren, deel 1* van ir. J.J. van Amstel e.a.

*Inleiding tot het programmeren, deel 2* van ir. J.J. van Amstel e.a.

*JSP - Jackson structureel programmeren* van Henk Jansen

*Aspecten van programmeertalen* van ir. J.J. van Amstel en ir. J.A.A.M. Poirters

*Programmeertalen, een inleiding* van ir. J.J. van Amstel e.a.

*Het Groot Pascal Spreukenboek* van H.F. Ledgerd, P.A. Nagin en J.F. Hueras

*Cursus Pascal* van prof.dr. A. van der Sluis en drs. C.A.C. Görts

*Cursus eenvoudig Pascal* van prof.dr. A. van der Sluis en drs. C.A.C. Görts

*Inleiding programmeren in Pascal* van C. van de Wijngaart

*BASIC, EIT-serie, deel 3*

*Cursus BASIC* van ir. R. Bloothoofd e.a.

*Cursus COBOL* van dr. A. Parkin

*Structuur en stijl in COBOL* van ir. E. Dürr en dr.ir. F. Mulder

*Cursus FORTRAN 77* van J.N.P. Hume en R.C. Holt

*Programmeren in LISP* van prof.dr. L.L. Steels

*Cursus ALGOL 60* van prof.dr. A. van der Sluis en drs. C.A.C. Görts

*Programmeren, deel 2: Van analyse tot algoritme* van prof.drs. C. Bron

*Inleiding programmeren en programmeertechnieken, EIT-serie, deel 1*

*Inleiding programmeren* van prof.dr. R.J. Lunbeck

## SYSTEEMPROGRAMMATUUR

*Bedrijfssystemen, EIT-serie, deel 4*

*Systeemprogrammatuur van drs. H. Alblas*

*Vertalerbouw van drs. H. Alblas e.a.*

*Unix, het standaard operating system van G.J.M. Austen en H.J. Thomassen*

## BESTANDSORGANISATIE/DATABASES

*Informatiestructuren, bestandsorganisatie en bestandsontwerp, EIT-serie, deel 5*

*Gegevensstructuren van R. Engmann e.a.*

*Bestandsorganisatie van prof.dr. R.J. Lunbeck en drs. F. Remmen*

*Databases van drs. F. Remmen*

## INFORMATIEANALYSE/SYSTEEMONTWERP

*Effectieve toepassingen van computers van M. Peltu*

*Vorbereiding van computertoepassingen van prof.dr. A.B. Frielink*

*Simulatie, een moderne methode van onderzoek van drs. S.K.T. Boersma en ir. T. Hoenderkamp*

*Systeemontwikkeling volgens SDM van H.B. Eilers*

*Een samenvatting van de System Development Methodology SDM van PANDATA*

*Cases op het gebied van administratieve organisatie en informatieverzorging (inclusief systeemontwerp) van prof.dr. P.G. Bosch en H.A. te Rijdt*

*Uitwerkingenboek bij Cases van prof.dr. P.G. Bosch en H.A. te Rijdt*

*Gegevensanalyse van R.P. Langerhorst*

*Evaluation of methods and techniques for the analysis, design and implementation of information systems, editors: J. Blank en M.J. Krijger*

*Analyse van de informatiebehoeften en de inhoudsbeschrijving van een databank van prof.dr. P.G. Bosch en ir. H.M. Heemskerk*

*Eerlijk en helder van prof.dr. P.G. Bosch*

## THEORETISCH/COMPUTERSCHAAK/TOEPASSINGEN

*Computers in de negentiger jaren van G.L. Simons*

*Expert systemen, ontwikkeling en gebruik van intelligente programma's van Henk de Swaan Arons en Peter van Lith*

*Abstracte automaten en grammatica's van prof.dr. A. Ollongren en ir. Th.P. van der Weide*

*De tekstmachine van dr. M. Boot en drs. H. Koppelaar*

*Computerschaak, schaakwereld en kunstmatige intelligentie van dr. H.J. van den Herik*

*Lineaire programmering als hulpmiddel bij de besluitvorming van prof.dr. S.W. Douma*

*Simulatie en sociale systemen, redaktie: J.L.A. Geurts en J.H.L. Oud*

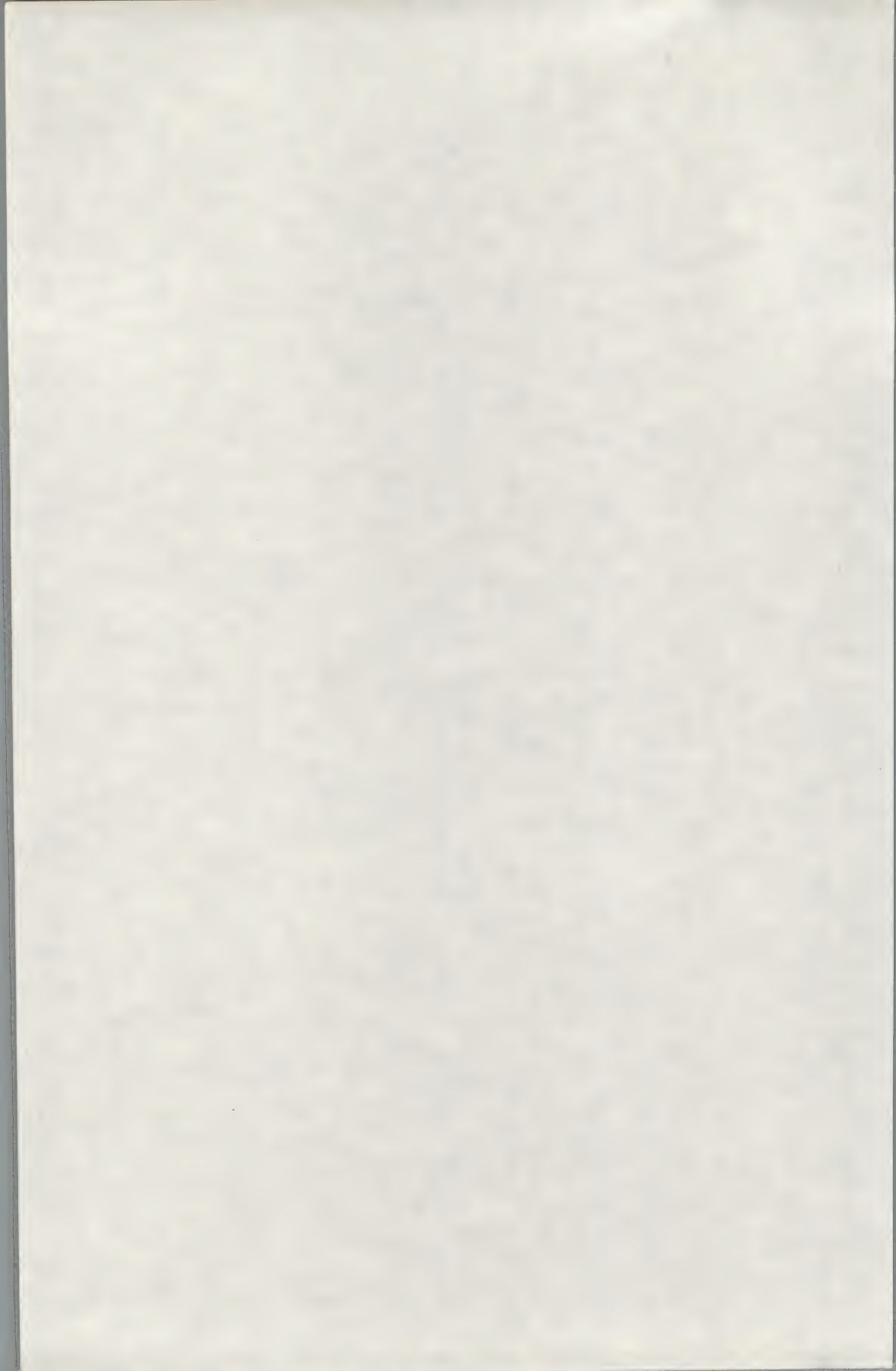
*Onderneming en overheid in systeem-dynamisch perspectief, redaktie: A.F.G. Hanken en J.H.L. Oud*

## INFORMATIE OVER DEZE PUBLIKATIES BIJ:

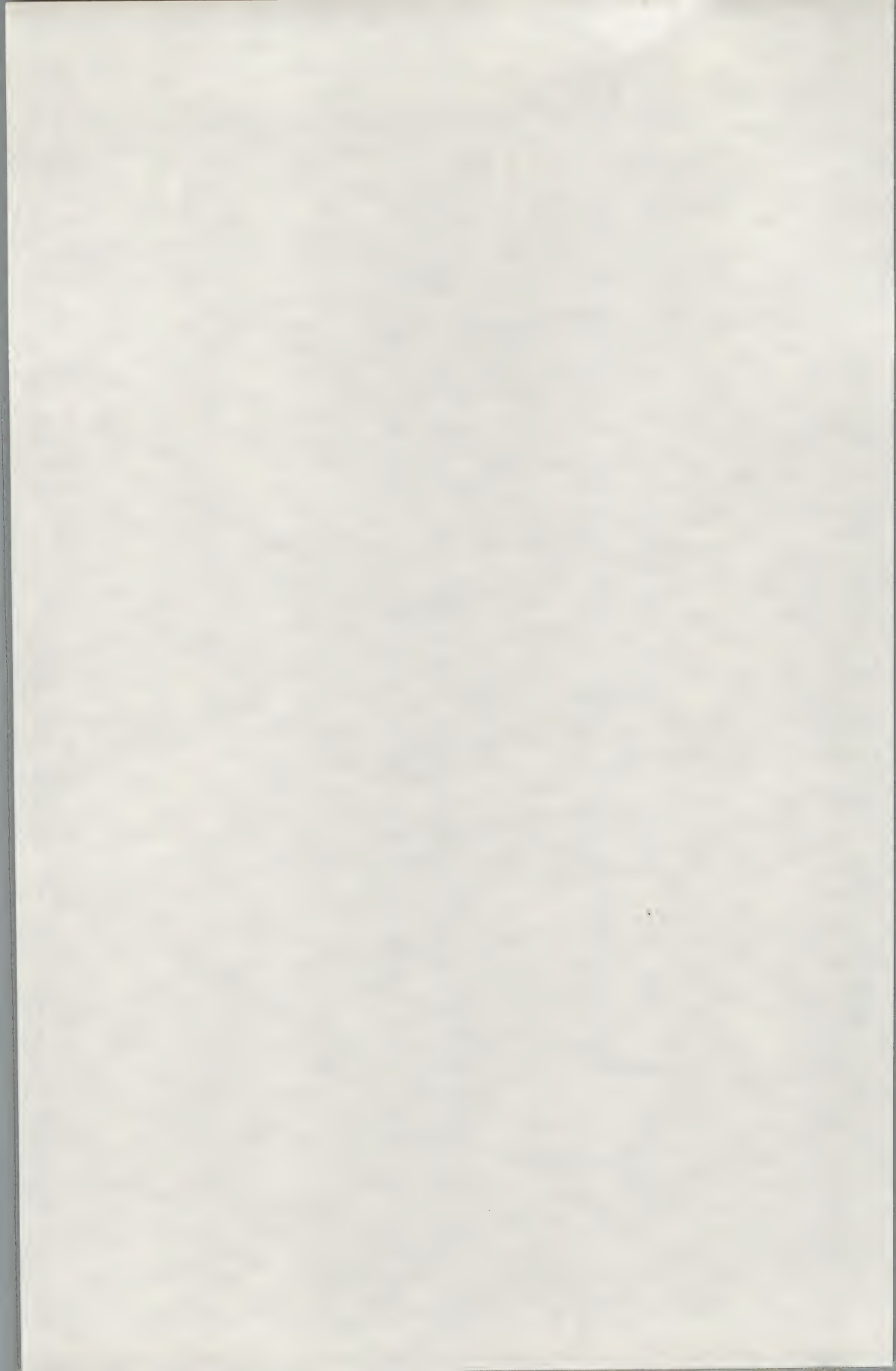
Academic Service, Postbus 96996, 2509 JJ den Haag

Tel.: 070-247238

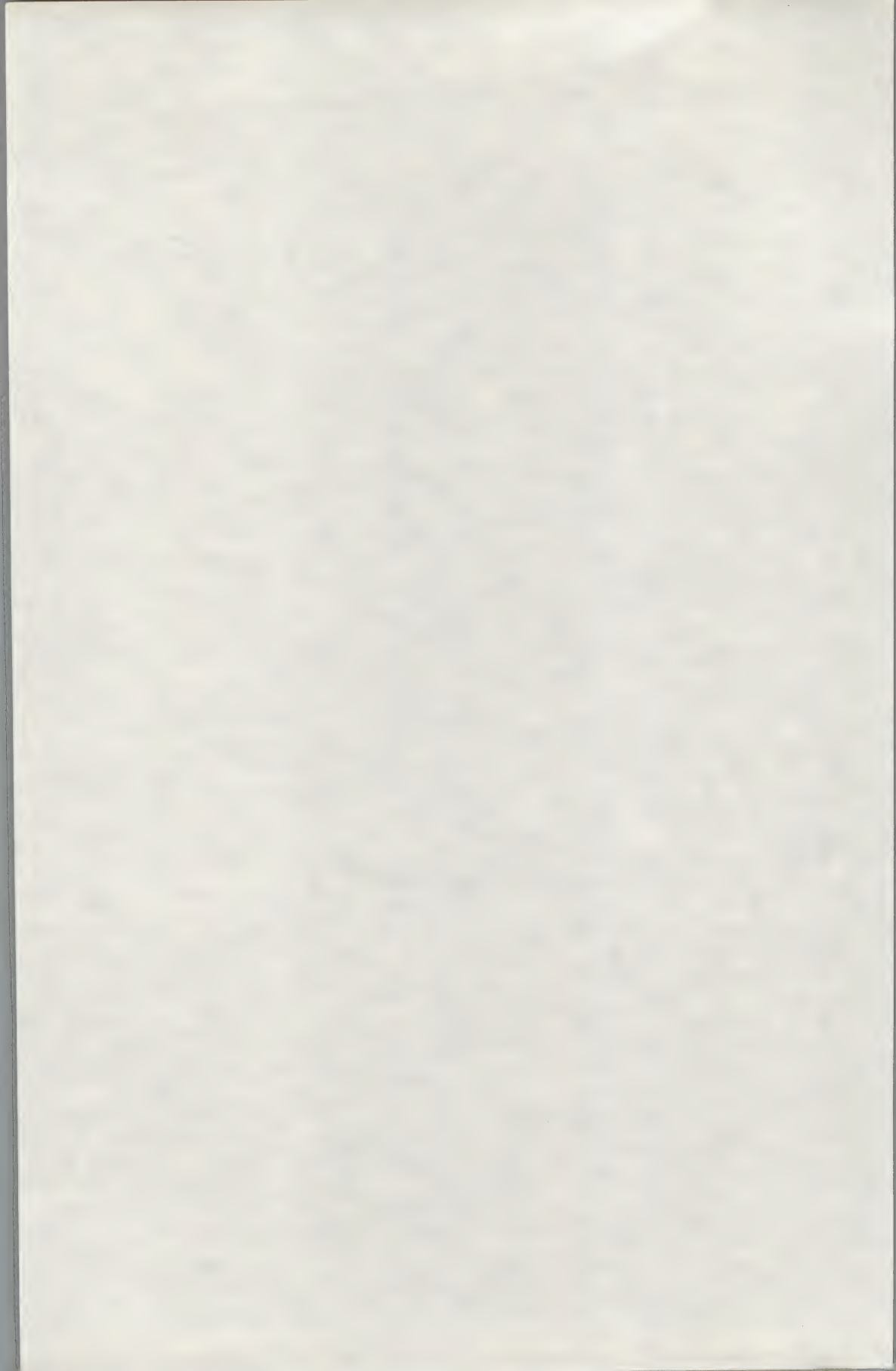




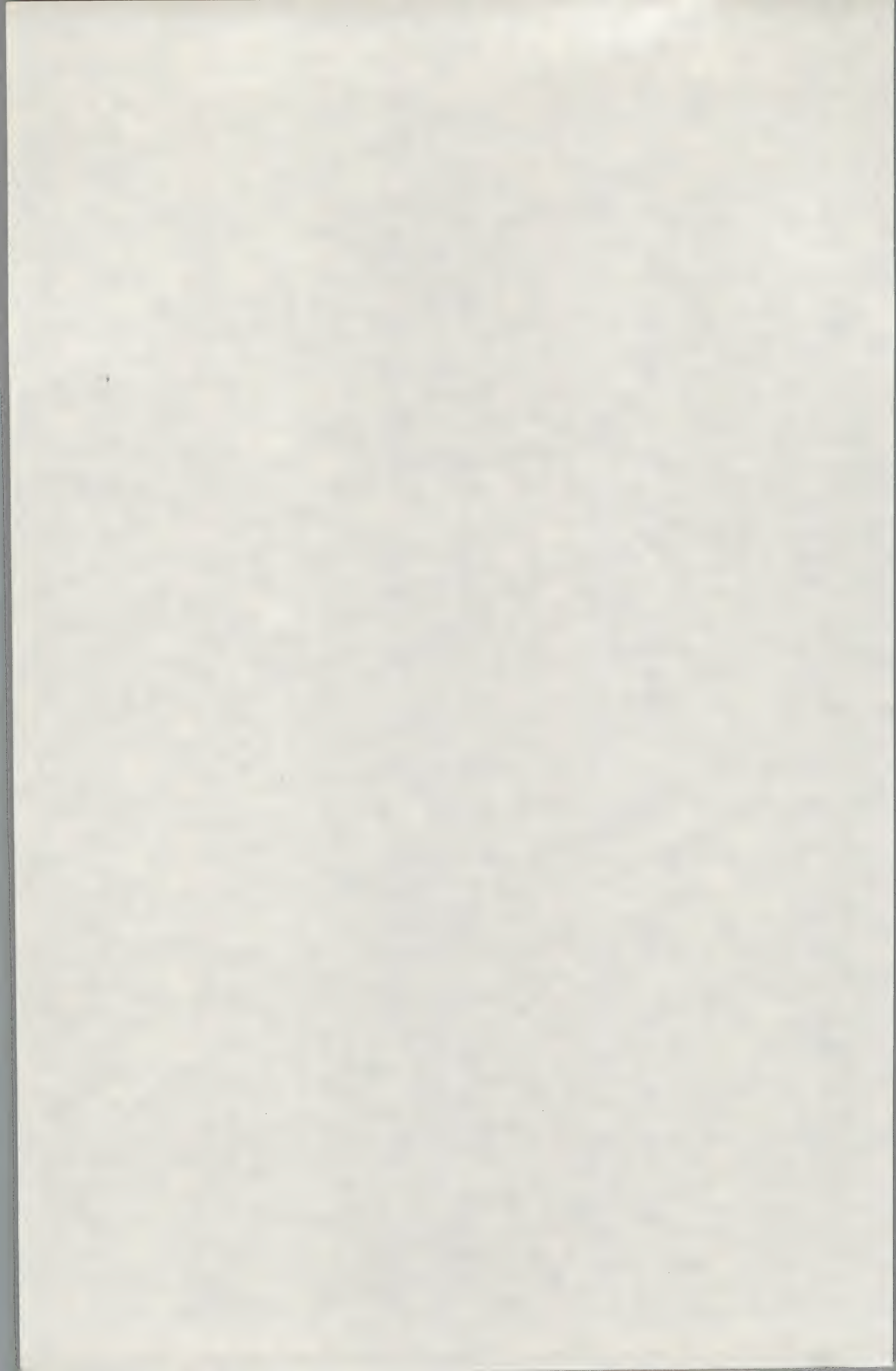




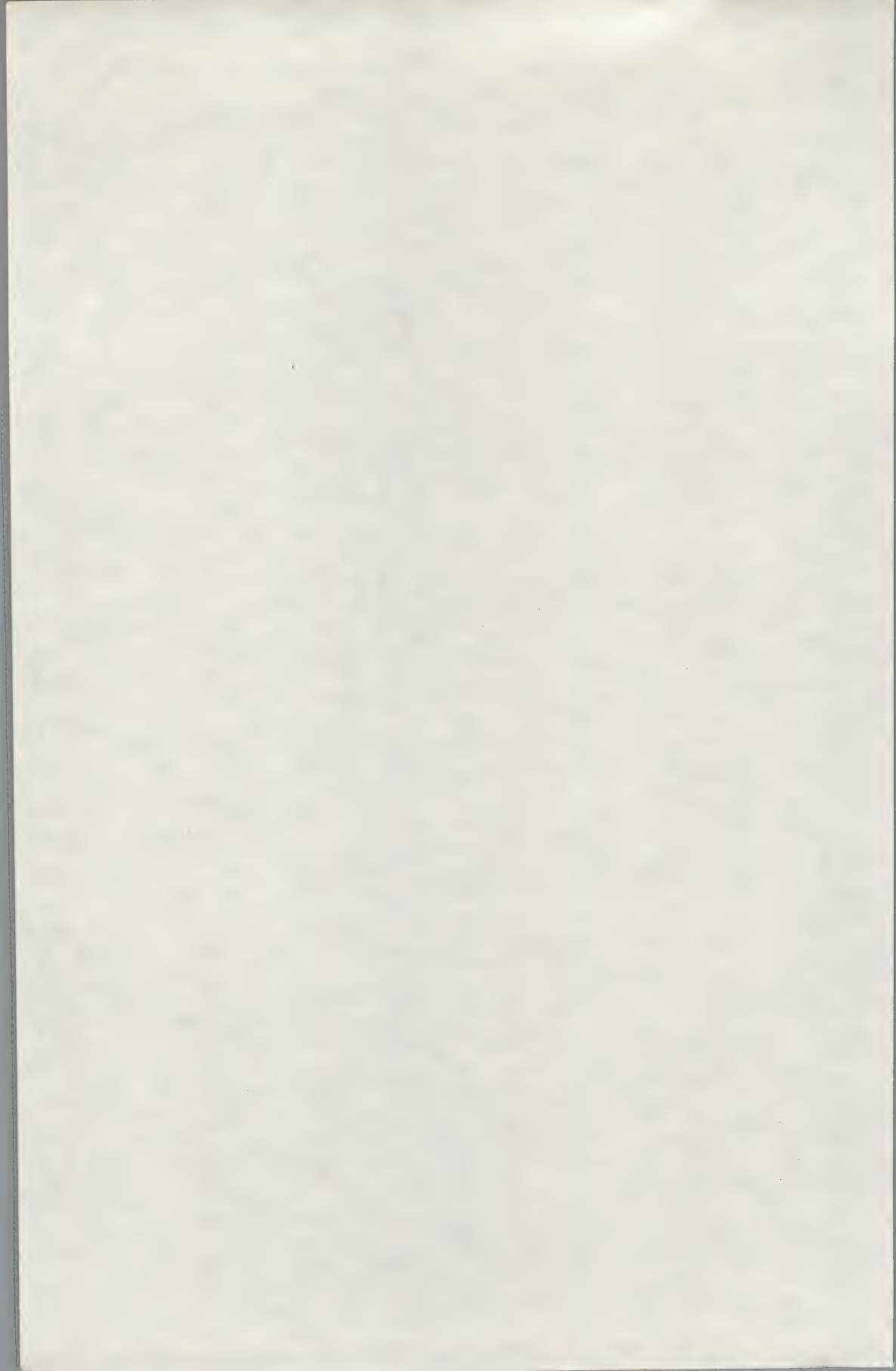


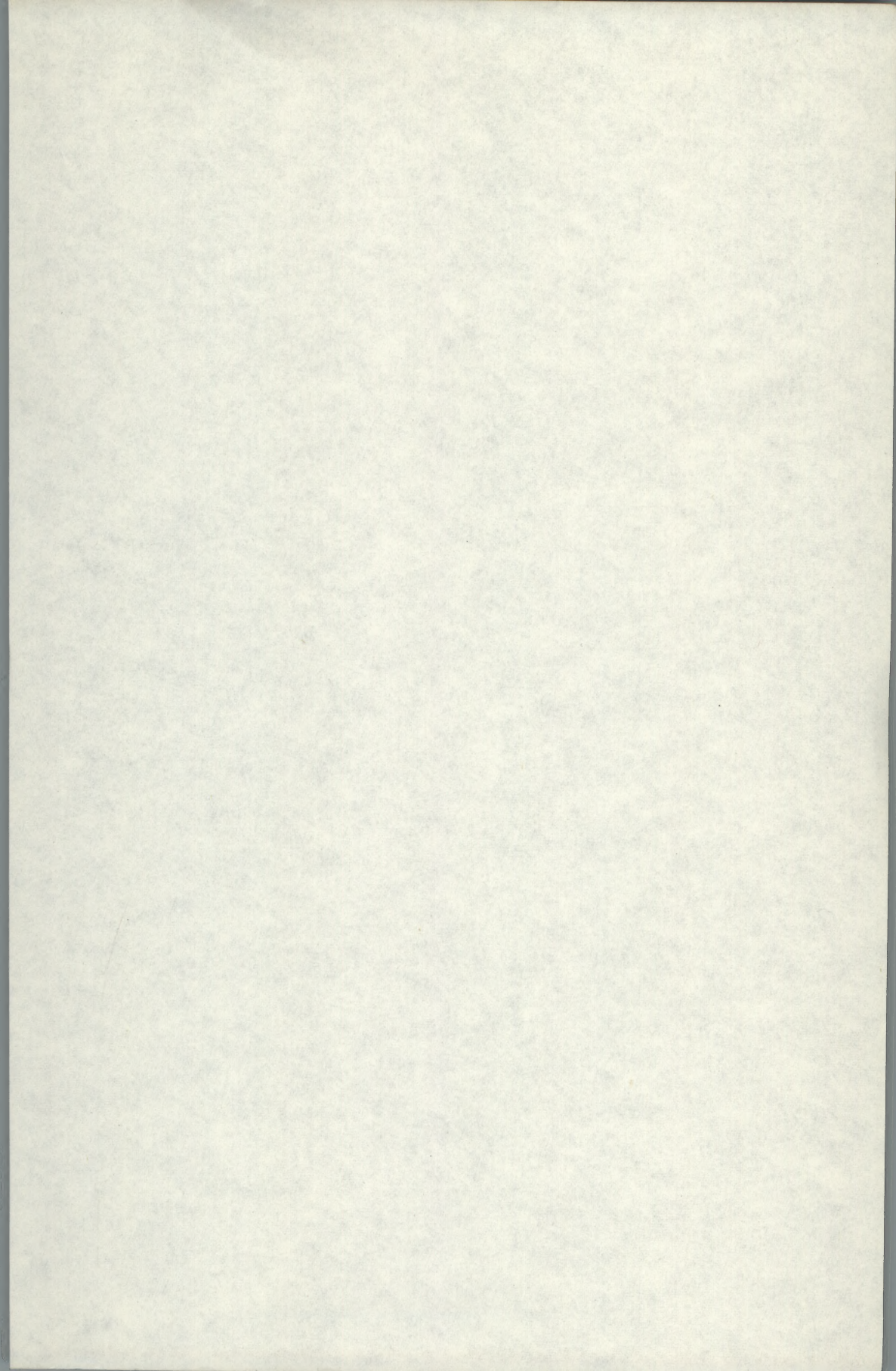


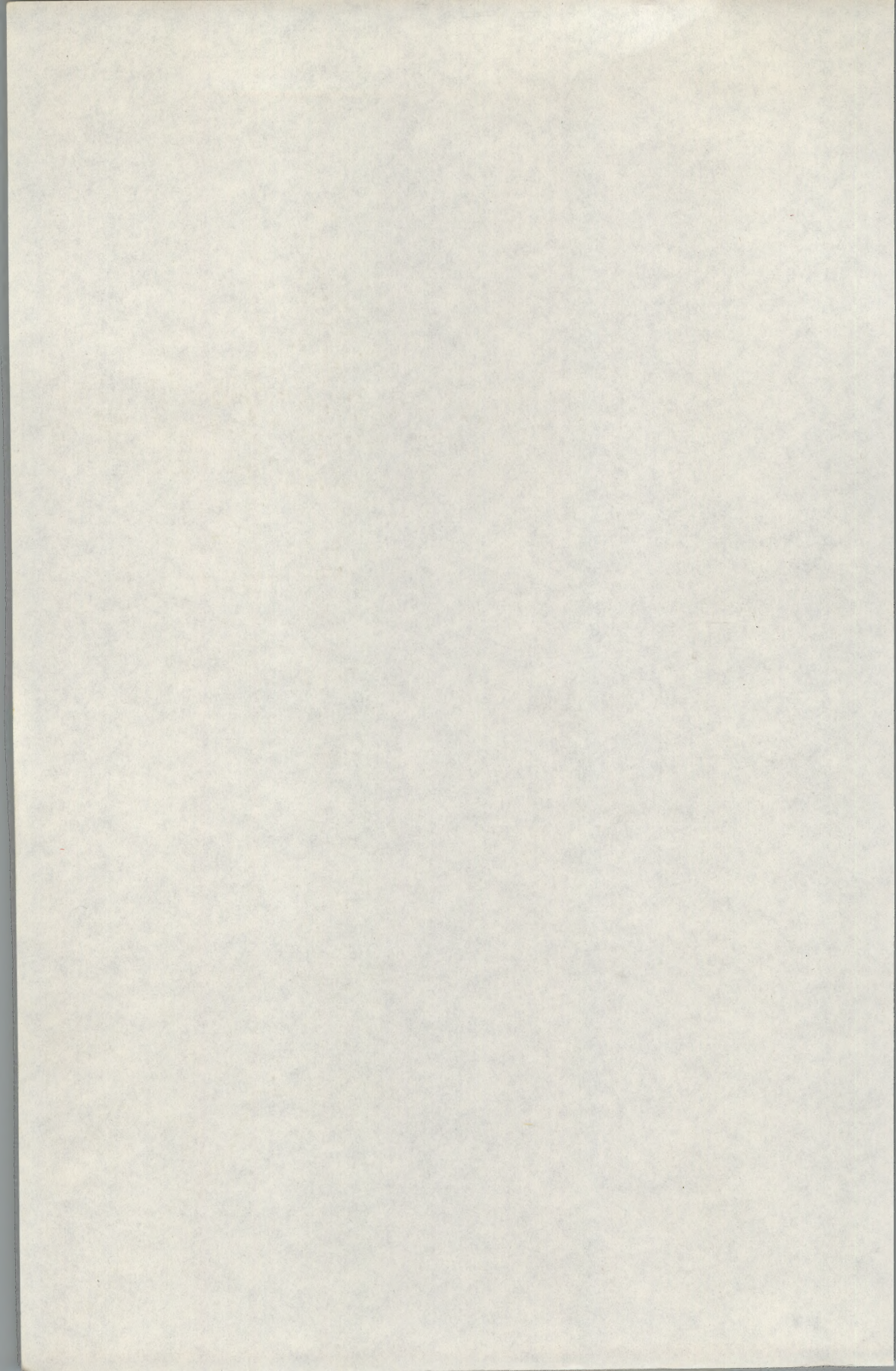


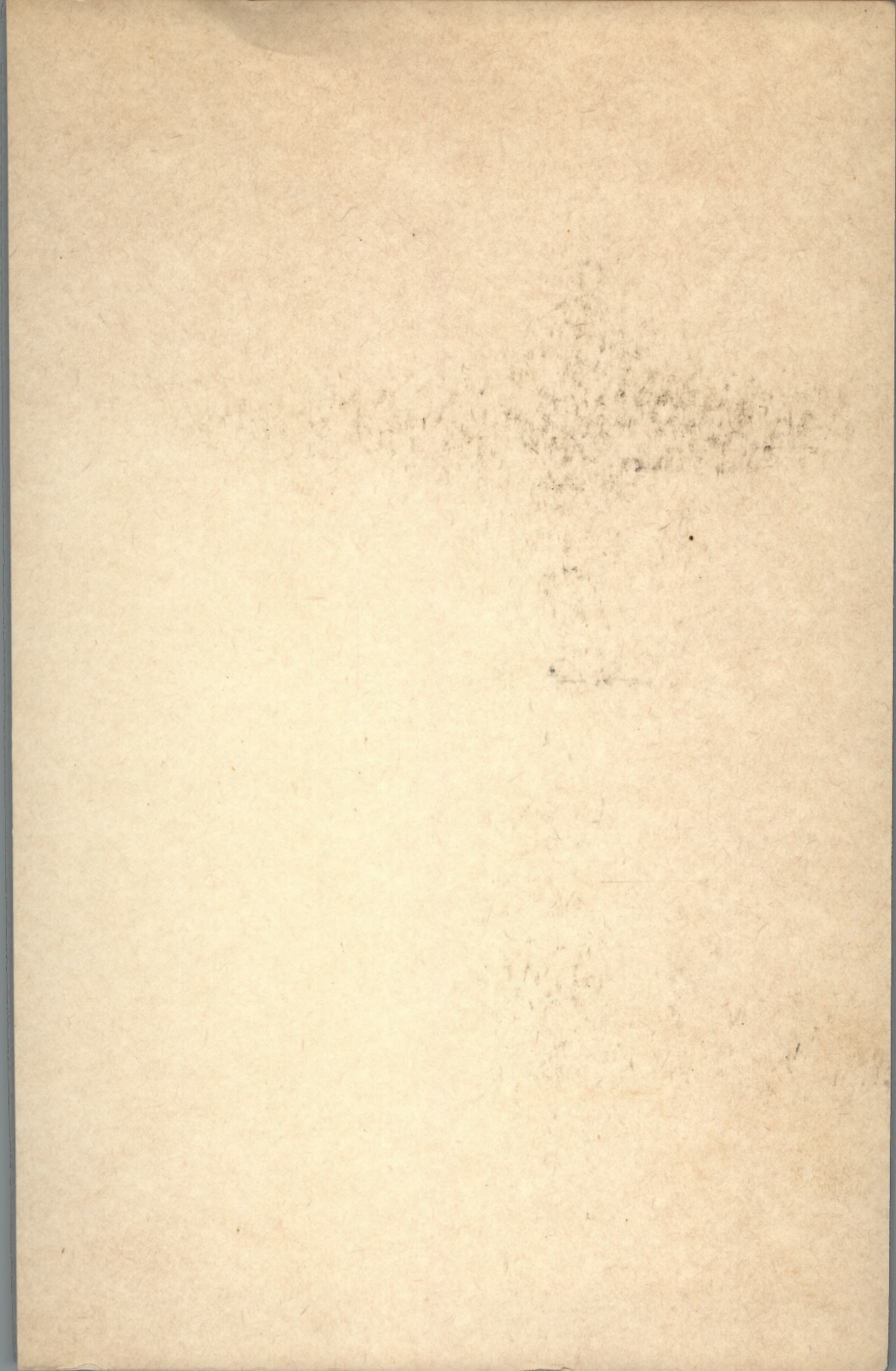






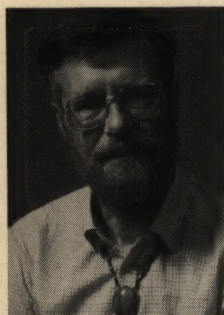






Dit boek behandelt het ontwerpen van correcte programma's, dat wil zeggen programma's die aantoonbaar aan een gegeven specificatie voldoen. Het bescheiden formalisme, dat hiervoor nodig is, wordt volledig geïntroduceerd, uitgelegd en geïllustreerd, zodat het boek met een minimum aan voorkennis zelfstandig kan worden bestudeerd. Doordat het programmeerprobleem volledig machine-onafhankelijk wordt benaderd, blijft de lezer verschoond van irrelevante details en is de methode algemeen toepasbaar en van blijvende praktische waarde. Het boek wordt afgesloten met een unieke serie opgaven, die hun waarde in de praktijk van het onderwijs hebben bewezen.

Prof. Dr. Edsger W. Dijkstra werkte van 1952 tot 1962 op de Rekenafdeling van het Mathematisch Centrum te Amsterdam, van 1962 tot 1984 bij de Onderafdeling der Wiskunde (en Informatica) van de Technische Hogeschool te Eindhoven en van 1973 tot 1984 tevens als Research Fellow bij Burroughs Corporation. Sinds 1984 bekleedt hij de Schlumberger Centennial Chair in Computer Sciences aan The University of Texas at Austin.



Ir. W.H.J. Feijen trad in 1970 als wiskundige in dienst van de Onderafdeling der Wiskunde van de Technische Hogeschool te Eindhoven, alwaar hij in de groep Fundamentele Programmering onder leiding van prof. dr. Edsger W. Dijkstra medewerking verleende aan de vorming van de Wiskunde van het Programmeren. Hij was verantwoordelijk voor de oprichting van diverse programmeerpractica. Sinds 1975 is hij als docent verbonden aan de Stichting Hogere Informatica.

